A 2D visual language for rapid 3D scene design

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Science

in the

University of Canterbury

by

Nathan Adams

Examining Committee

Dr. R. Mukundan

Dr. Burkhard Wuensche

University of Canterbury

2009

Abstract

Automatic recognition and digitization of the features found in raster images of 2D topographic maps has a long research history. Very little such work has focused on creating and working with alternatives to the classic isoline-based topographic map.This thesis presents a system that generates 3D scenes from a 2D diagram format designed for user friendliness; with more geometric expressiveness and lower ink usage than classic topographic maps.

This thesis explains the rationale for and the structure of the system, and the difficulties encountered in constructing it. It then describes a user study to evaluate the language and the usability of its various features, and draws future research directions from it.

Table of Contents

Acknowledgments

I would like to thank my supervisor, Dr. R. Mukundan, for his invaluable support, advice and encouragement throughout the length of this project, which ultimately made it possible. Thanks is also due to my co-supervisor, Dr. Richard Green for his enthusiastic input and suggestions.

I must also acknowledge the support of many of my family and friends, without some of whom it is also unlikely that this would have been possible. Particularly I thank Cameron Oliver, whose review and suggestions added noticeably to the coherence of this work.

Chapter I

Introduction

The computer-aided extraction of data from 2D raster maps (such as those scanned from paper sources) designed for human readability is a large and general problem with a wide range of ongoing research ([1], [2]). Since many such flat-projection maps contain various types of topographic data encoded in them, one possibility afforded by a successful digization of such maps is the reconstruction of a 3D landscape or scene based on what such data can be found within them. The standard means of encoding the height of terrain on a map is with contour lines. But other methods become necessary if we want to represent a 3D scene more complicated than just open terrain. In the case of buildings for example, conventional buildings are represented with (at most) a silohuette of their planimetric outline. Without any further information, a 3D re-creation is unable to do anything more than extrude the silohuettes vertically to an arbitrary height.

This problem can be framed with an example application that could benefit from map digitization techniques that recognise such alternative representations of height data. The production of 3D computer games is often heavily dependent

on 3D scene designs ('levels' or 'maps') as fundamental art assets. Due to increasing art content design costs, it has become recognised that best practice in the game design process should involve an early period of heavy design iteration, so that costly adjustments can be avoided later, or so that developers do not find themselves 'locked' into an unsatisfactory design [3].

1.1 Rapid prototyping

Pen-based interface research has frequently focused on rapid prototyping and design applications as the sort of application that stands most to benefit from such interfaces.

[4] describes the iterative design process for one major game title, and how level designs began as diagram sketches to then be passed to a level designer, who would manually construct a prototype level with very basic design elements from them. The prototype would be play-tested and if found unsatisfactory, iterated upon.

As a consequence of this need for rapid and iterative design testing in the game design process, there is much attraction in the idea of a system that supports this process by directly (or as nearly as possible) translating designer sketches (i.e., a 2D raster map, either scanned from paper or sourced from the designer directly drawing on a tablet device) into a testable prototype. [5]. [6] developed such a system for laying out 3D maps based on a top-down sketch of a scene. However,

no method for encoding height data was developed, although the possibility of a mode that would interpret isometric (i.e, axonometric) input was left open for future work. As games often require a combination of outdoor, terrain-dominated areas and more enclosed and artifical structures (i.e, that often require alternate depiction methods), such a system would have to be flexible enough to allow a combination of styles to be used to construct a scene. This in itself is a design challenge.

1.2 A paper-based system

It is important to carefully distinguish different types of pen-based input systems, and identify their significant differences and commonalities.

Shilman used a taxonomy that classified "pen-based tools for design and creativity" along the dimensions of *sophistication*, *deferral*, and *flexibility* [7]. Sophistication was the measure of how detailed and structured the user input could be. Deferral measured how soon the interpreted input would be revealed to the user, and flexibility measured the breadth of domains a system supports.

From the perspective of the system designer, these dimensions are not fully orthogonal; there are correlations between the three that aren't fully obvious without extending the taxnomy. For example, a huge advantage in the implementation of sketching systems is given if pen movements can be captured, skipping the troublesome process of stroke differentiation. This can make for significant difference

in what level of sophistication a system can feasibly attempt to support. This can be seen in a performance comparision of online and offline handwriting recognition systems; the availability of penstroke velocities gives much better results than post-hoc recognition of a raster image. [8]

From the end-users' viewpoint this distinction should not matter, as long as the final result is the same. To capture this differentiation Shilman's taxonomy can be extended in two ways. The first is to distinguish between *online* and *offline* systems: those that capture pen motion as it happens, and those that capture the output after the fact. Pragmatically this difference is strongly related (but not completely overlapping) to the second differentiating point: those systems that rely on raster input and those that rely on vectors (which may or may not include stroke orderings).

The vast majority of systems fall on the same side of both of these two divides of when input is captured and in what form: a tablet PC provides immediate stroke input to the system in the form of vectorized stylus movements, while a scanned paper or photographed whiteboard diagram provides all the user's input simultaneously to the system as a raster image. For this reason when a system is hence referred to as online or offline without clarification it should be also read as implying a vector input or raster input, respectively. This is only due to the prevalence of most systems that fit this pattern: a system like the Digital Desk [9] that performed online image analysis of the user's workspace at a desk as captured

by a live video camera is an exemplar of an exception to this pattern.

Online sketching systems offer many advantages that ease the recognition problem, such as making it relatively simple to differentiate intersecting strokes. However, systems that require an online presence or even offline but stroke based input as opposed to a raster image forgo some flexible input options, such as images of a whiteboard or scanned ink on paper.

It is worth noting however that good attempts at crossing the divide between paper and more digital input methods have been made with augmented paper systems such as PapierCraft [10] and ButterflyNet [11], both based on the Anoto digital pen [12], a pen with an inbuilt camera that tracks the nib's position using a special dot pattern preprinted on the paper. Such systems provide a possible strategy and direction for moving a purely offline system such as the one described in this thesis towards an online model without forgoing any existing functionality.

An offline, raster image based system like the one that will be described maintains an inherent forward flexibility. In supporting pen and paper or whiteboards, it can provide the cheapest and most versatile input solutions. No digital pen and special paper, tablet PC or stylus is required. Yet it can be augmented with such technology. A video camera (or just a still camera taking shots at regular intervals) watching a whiteboard could provide semi-live updates of a design as it appears incrementally on the board. A digital pen interface would allow for the tactile and other physical advantages of paper while allowing incremental and live display of

the interpreted diagram as it is sketched.

1.3 Topography in paper flatlands

While there is much interest in sketch-based design tools for prototyping 3D models, how to implement this in an offline system has been relatively unexplored, compared to the emphasis on online systems.

Despite their free-form flexibility, there are downsides to freeform sketches as an expressive medium for 3D geometry. They do require some artistic talent for a user to be confident in drawing in a 3D perspective. But even for a skilled user, there are classes of geometry that are not well expressed in a conventional free-hand sketch. Open landscape particularly is hard to sketch visually, as by its nature sketching is adept at capturing the siloheuttes and defining lines of an object, while hills, ravines, and other such contoured surfaces do not afford easy expression in sketch form. Curved forms are often expressed in sketching with shading, but this increases the artistic ability required of the user if it is to be done at all well.

The 2D expression of such 3D geometry has been a problem that somewhat naturally fell into the lap of cartographers throughout history. The solution now usually used is the isoline or contour line, a contiguous line that runs through all the points at the same elevation on a surface. But it has not always been the default expressive mode for topographic maps. One approach commonly used

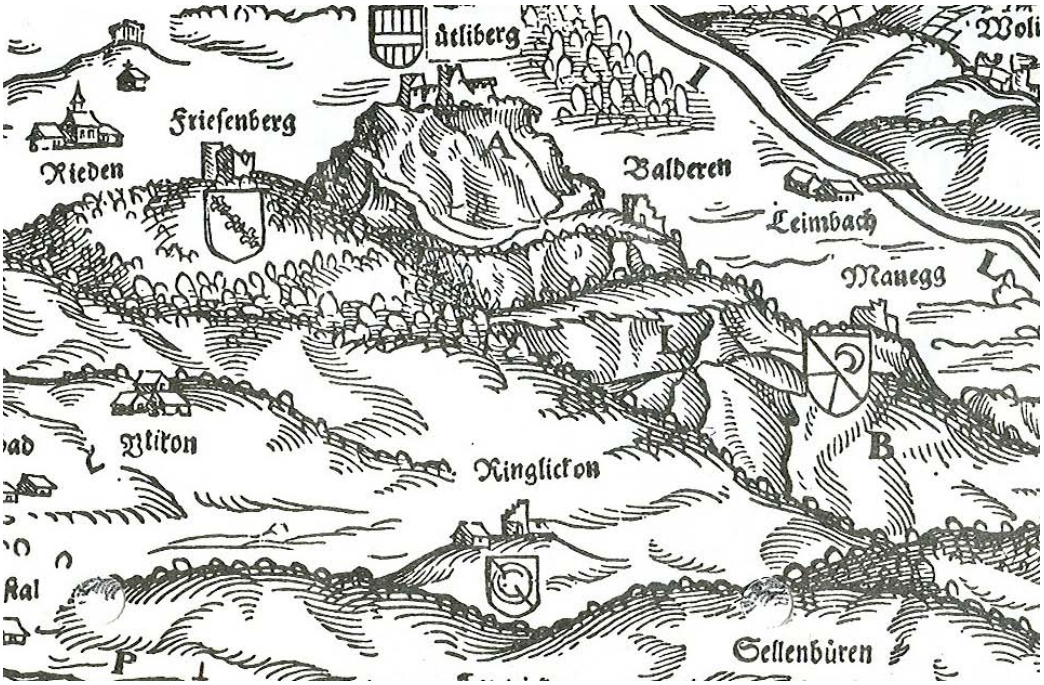


Figure 1.1: Zürich Canton, by Jost Murer. 1566. [13]

on early maps was to draw items of topographic note such as hills in a profile view, sometimes little more than an iconic depiction, others more graphic with an oblique projection and well integrated into the surrounding map, such as in Fig. 1.1.

Fig. 1.2 shows an example of a topographic map illustrated with *hachures*, lines that run down the slope instead of along it like contour lines. While still making for a less accurate depiction than contours since it cannot convey absolute altitude but only a relative difference between two areas, the hachure line does have a more natural 'reading' to it. Crone observed that

> *Hachuring, however, has several defects; if carried out elaborately, the*

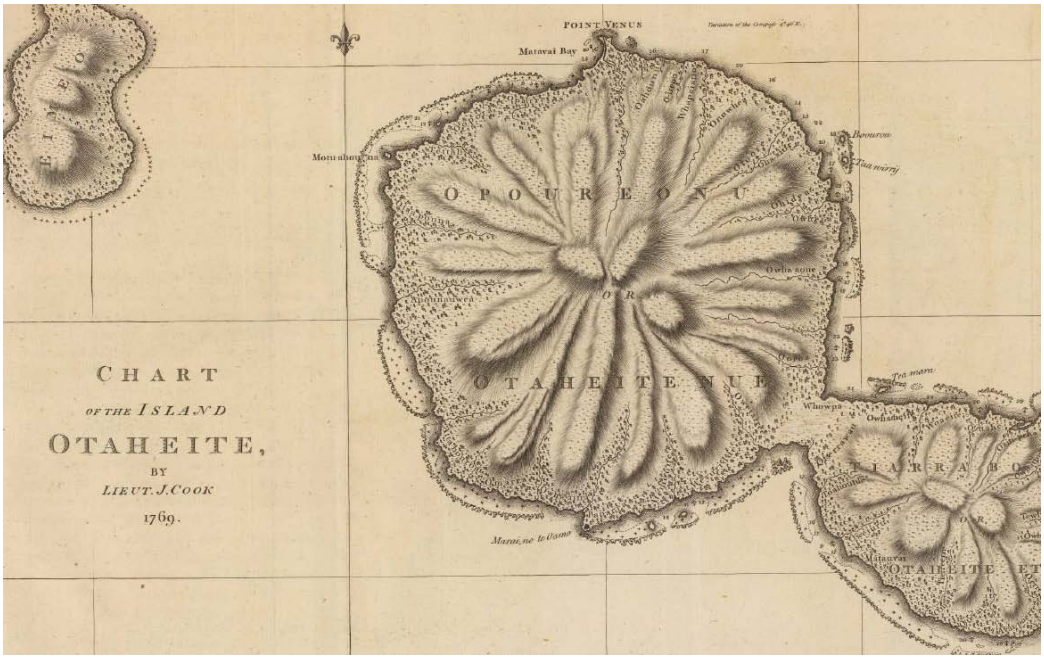


Figure 1.2: Otaheite (Tahiti), by the then Lieut. James Cook. 1769. [14]

> *heavy shading obscures much of the other detail on the map, and by itself can give no absolute value for the difference in elevation between one point and another.*
>
> *The value of contouring lies in the fact that, unlike hachuring, it enables the altitude of a particular point to be determined with considerable accuracy . . . It does not, however, always permit an idea of the relief to be formed rapidly . . .* [15]

Though effectively made redundant for any practical use in contemporary topographic map by the contour line, these two techniques of both profile images and using lines longitudinal to a slope to describe topography suggest there are

other useful approaches to a visual language aimed at capturing 3D geometry in a paper flat land than just contour lines. This thesis describes such a language that attempts to maintain or improve on the expressiveness of contour lines while reducing the effort required to use it, and the implementation of an automatic (or near-automatic) system to translate it into 3D mesh geometry.

1.4 Thesis layout

Chapter 2 briefly surveys some of the previous work into such systems, and related systems in domains such as sketch based design and topographic map recognition and vectorization. In chapter 3 the design goals outlined in the previous paragraph are expanded, elaborated and justified. This provides a background for the decisions shown in chapter 4 as the language itself is laid out and described with examples. In chapter 5 the exact structure of the system is described and illustrated by an explanation of its implementation, along with some rudimentary performance analysis and discussion. The more important evaluative question of whether the system is usable is addressed in chapter 6 with a qualitative user study. Finally chapter 7 concludes the thesis, summarising its content and discussing potential for future work leading from it.

Chapter II

Background

The system outlined in this thesis draws on several related yet distinct fields of research. The first is that of sketch and ink-based interface using systems, specifically those developed for design and prototyping work. The second, visual language recognition and support, is more specific and partially derives from the need created by this first area for robust means of sketch and ink recognition. Finally the field of topographic (i.e, contour) map vectorization and recognition is considered, as systems to process and understand raster images of contour maps and the visual language they employ is an area into which much work has gone.

2.1 Sketch and ink-based design tools

DENIM was a sketch-based website design interface produced as the result of an ethnographic study of professional web designers and observation of their design process [16]. The authors noted that such sketch-based tools (i.e, online recognition) could not completely replace paper in such a role due, citing its cost, its portability and its feel as advantages inherent to the medium.

2.2 Generalised visual language platforms

Such efforts have largely remained at a level specific to the domain of topographic maps. The encompassing problem of visual language recognition has been investigated at a higher level than such attempts, usually focussed on the goal of producing a system that given a grammar for a visual language, can then produce good recognition results. Ideally then, such systems could be used in recognising diagrams from domains as varied as circuit diagrams, graphical user interface mock-ups, family trees, etc.

One attempt to produce a domain-independent sketch recognition system was LADDER [17].

2.3 Topographic map vectorization

A large body of techniques has been developed for the vectorization of general planimetric and topographic 2D maps. Vectorization encompasses the segmentation — and, wherever possible — the semantic classification of all relevant features in a raster map. Feature relevance is task-dependent: identifying road intersections ([18]) and forest symbols ([19]) are two significantly different tasks, but as [19] found, for small features the best approach may involve the identification of all other features on the map anyway, so that they might be eliminated from the pool of possible matches.

The applied task we will be examining these techniques in is that of building a

3D visualization of the target map. The framework was designed in the context of rapid scene development for 3D games and the like. Therefore, the relevant raster map features for us will be those that could be of most use to a game designer rapidly laying out a game level prototype.

2.3.1 Text separation

In most map vectorizing systems, one of the first classes of image feature to be identified for removal is text. In the majority of maps the task of text identification is decidedly more difficult than standard documents, due to the density of raster image features and the looser constraints on text character rotation. As we will be able to have some *a priori* control of the input map, being able to lay out the system's formatting rules we may be able to minimize this problem in our framework. However, this would be an artificial limitation to impose, so preferably we will be able to avoid it. An alternative is to mandate the user use a unique ink colour for text, so that colour segmentation could be much more robust. This is not an unprecedented approach. [20]'s PROMAP system used pre-defined colouring to segment separate data layers, particularly to distinguish text from other features.

Early attempts at mixed text/graphic separation met with some success at identifying text with varying fonts and rotations [21][22]. These used connected-component searches, trying to identify the lines that strings of characters followed and then examining their proximity in those directions for adjoining characters.

However, these approaches failed in situations where the characters touched another graphic, or were overlaid on too varied a background. These problems are not trivially solved [23], though many attempts have been made ([24][25]), using different techniques of colour segmentation [26], adaptive filters [27], and knowledge-based heuristic approaches, such as the algorithm described in [28] which was based on the observation that text characters have uniformly short strokes compared to the line segments in many other map features. Other heuristic approaches use the semantics of the text itself to help recognition. [29] used a knowledge base of likely terms (a gazetteer) to make forecasts about adjacent text. [30] built an algorithm on four sequential levels of heuristics: first the semiological of connected component analysis to join pixels into characters, then stringing characters together, then detecting related characters, and finally a natural language heuristic operating at a semantic level.

2.3.2 Contour Lines

The most obvious and common class of height indicator in many 2D maps is the contour line, and the altitude markings that usually accompany them (though vectorisation is still possible if the altitude labels are sparse [31]). They provide a simple and robust system, and are often of high enough quality to use in a digital terrain model (DTM) if vectorization is feasible. For example, the official Switzerland DTM is based on its 1:25000 scale topographic map rather than a

new geographical survey ([32] via [33]). Because of the wealth of topographic information stored in such paper contour maps, the value of automatic vectorization was evident early on, and has been an active research topic since the late 1960s (c.f [34],[35]). The consequence of this is that although research on this specific problem continues, a mature body of techniques is available to us. Various algorithms have been developed for each stage, but [36] noted that most use the same basic clean-up and extraction process shown in [37] and [38] of filtering, thresholding, thinning (often by erosion filters), followed by the vectorization of the remaining lines.

Once contour lines are extracted, typically a follow-up step is to interpolate the height values for all points on the terrain's surface not explicitly covered by contour lines [39], and since we will be transforming our maps to 3D geometry, conversion to triangles will be necessary [40][41].

2.3.3 Building extraction

It has been noted ([42]) that on a topographic map, building outlines (if present) can be detected by the geometric features of the line segments that compose them, provided that the same segmentation, filtering, and thresholding necessary for contour lines can work. They will be of a standard width, be relatively straight, with reasonably predictable bounds of angular change between each segment, and most importantly, they will form a loop.

2.3.4 Semantic symbols

Of considerable importance to a designer trying to embed meaning into a map, or the program trying to parse it, are the semantically significant symbols available for use. As we are designing a framework the user will need to annotate liberally with special markers (for example, indicating at what position a player starts off in a level), semantic symbol recognition is an important aspect.

Conventional maps explain their semantic symbols through use of a legend or key. [43] and [44] describe a system that uses legend tiles as a training set to extract shape descriptors. There is an appeal for us in using such an approach, as a successful implementation would allow for the user to *create* their own legend, so that verbose annotations may be attached to small symbols without the clutter of joining arrows. Similar systems have been implemented for architectural floor-plan recognition. [45] used a neural network trained to recognise architectural symbols by a series of constraints described in a shape grammar, but it required the constraints to be pre-supplied from a description file, tediously building each symbols up from primitives of arcs, line segments, and angles. [46] described a rule acquisition system that required one sample of a symbol to begin recognition, and would improve continually as more symbols were identified, basing its rules on local shape variants. [47] proposed an error-tolerant isomorphism graph matching model capable of handling breaks in the symbol lines, thus making it a better candidate for recognising hand-drawn symbols. The ten year retrospective

[48] noted that "although there have been a number of successful complete symbol recognition systems, these are mostly within areas with relatively few kinds of symbols to discriminate and within areas where it is easy to localize or segment potential symbols."

2.3.5 General integration

As the various techniques available for raster map conversion have become more well-defined and understood, effort has also been focused on the problem of how to unify these techniques into a more general framework. [30] noted how due to the feature density of the maps they were attempting to vectorize, they had to adopt an "onion peeling" strategy; a progressive simplification of the target map by starting with the easiest to identify layers of information, and then removing them from the map once identified. This is a common approach.

This approach is typical for the problem: a consecutive pipeline of filters, extractions and transformations, all tuned to the style of the target maps. Some such as [49] and [50] made one or more of these steps interactive to improve accuracy, but it is usually desirable to minimize the need for human intervention as much as possible. Such a pipeline often follows a general pattern, combining steps such as colour segmentation, line detection and connecting, text detection and removal, semantic symbol identification, and higher-level techniques, depending on the approach and map type.

Some techniques have benefited from repeating or parallelizing steps in the pipeline ([51],[52]). This approach can soften the disjointing effect the removal of entire data "layers" can have, and allow for expert knowledge to guide the intermediate data in a sensible direction. This can be taken further into a genuinely co-operative process where two or more stages of the pipeline feed into each other until some qualifier heuristic is satisfied ([20],[53]). [53] took the approach of iteratively refining the pipeline's product (a road network) with a collection of heuristic knowledge about the shape of roads, allowing road fragments that had been disconnected by the presence of other map features to be rejoined. The first part of the system however still followed a fairly standard semi-linear pipeline.

Chapter III

Design Objectives

This chapter outlines the desired goals the system and language were aimed at fulfilling in their design and implementation.

3.1 An offline language

As outlined in §1.2, a distinction needs to be made between sketch systems (and the visual languages they support) that operate on an online or an offline basis. Offline systems support a wider range of inputs including raster images of diagrams, but this usually comes at a trade-off of sophistication. If offline support is to be possible, then the language needs to be designed with this in mind, so that parsing without any information about pen strokes is made feasible.

From previous literature and multiple examples it can be seen that differentiation of distinct information layers in a 2D map is most commonly and robustly solved with the use of distinct colours. While this is a major aid to computer-based feature recognition and distinction, it is also desirable from the human user's perspective as a means of decluttering an information-dense image[54]. It would follow that encoding a semantic meaning to different ink colours is a sensible

design feature (and accomodation, as enforced colour-based layer differentiation does make the system's task simpler).

Recognising that no system of 2D diagrams can properly capture the full range of possible 3D geometry with a single view, it is also prudent to consider what types of 3D 'mesh' should receive the most language support. As the universal use of contour maps for most sort of terrain maps for the past several hundred years shows, it is possible for certain classes or domains of topology, such as terrain, to be well-described by a visual depiction designed for that domain.

This system focuses on a sample target application of a rapid 3D game level prototyping environment, and from this comes a desire to focus on map characteristics typical of many 3D games: alternating outdoor terrain (the sort often well captured by traditional topographic maps with contour lines) with artificial geometry with more vertical and horizontal flat surfaces, particularly typified by buildings. Contour maps are a well-understood medium for topographic description; have chosen to focus on improving on their short-comings without removing any expressiveness.

3.2 Saving effort and ink

One pragmatic aspect that distinguishes this language from that used in contour maps is that the latter is designed almost exclusively with a focus on serving the audience side of the creator/audience balance. In a system where the map author is

the primary user of the language (as the whole aim of the systems is that maps and diagrams will be translated into a 3D model which will be the primary viewing medium for the audience), language design must focus on easing the job of the creator. This implies an easing of both mental and physical effort on the creator's part.

Minimizing cognitive load then requires that the basic forms of the language — or to borrow a term from the domain of programming languages, its primitives — reflect a close mapping to how the map creator actually thinks about a landscape or other object to be modeled, yet at the same time not require more than a modest level of artistic ability.

Physical effort in using such a system can be effectively measured by how long the user takes to draw a map. This has several effective practical requirements in the context of an offline system: the foremost one being that the language forms' expressiveness minimize the amount of ink use required. If the language requires that different ink colours should be used, then the number of inks should be minimized to reduce the time overhead of switching pens.

3.3 No less than contours

As a base-line of expressiveness, the language should provide a superset of geometric expression to that provided by traditional contour maps. Since the target domain is not to provide accurate terrain relief depiction, it is acceptable if the

language requires more effort or ink than contour maps to model the same geometry, provided that it allows for domain-useful geometry that is not possible with contour mapping. An example of this would be vertical surfaces: an infinite gradient is impossible to represent on a contour map, yet many artifical surfaces particularly such as buildings and walls require them.

3.4 Scale

The language should provide some sort of construct or mechanism to allow a flexible scope of scale, so that geometry can be described to a reasonable level of detail without compromising the size or detail of the surrounding context.

3.5 Reuse

Due to the time savings (both in production and in end use tasks such as rendering) that replication and reuse of existing assets provides, the language should allow for the reuse of pre-made geometry, whether it comes from the user or another source. Ideally, once a component has been used in another, it should still be possible for the original component to be modified or replaced and the updated geometry correctly appear in the places it has been referenced.

Chapter IV

Language description

In deciding on the language's "grammar", the design objectives outlined in the preceding chapter were used as a criteria for choosing features. Concisely, the aim of the feature set is to provide intuitive end results for a user, while minimizing the amount of drawing required. This would be best supported by allowing the reuse of geometry (whether user-created or from third-party sources). Finally, maximising geometric expressiveness is desirable.

4.1 Closed regions and height markers

The language is centered around the description of geometric space as a set of planimetric closed regions. In Fig. 4.1.A there are three such closed regions, drawn in black ink. Inside them in red ink are written three height markers. The closed regions as a whole are assigned a height value by their corresponding marker: single-letter markers correspond to steadily incrementing heights in a look-up dictionary ($A = 100$,$B = 200$,$C = 300$ etc). The corresponding output

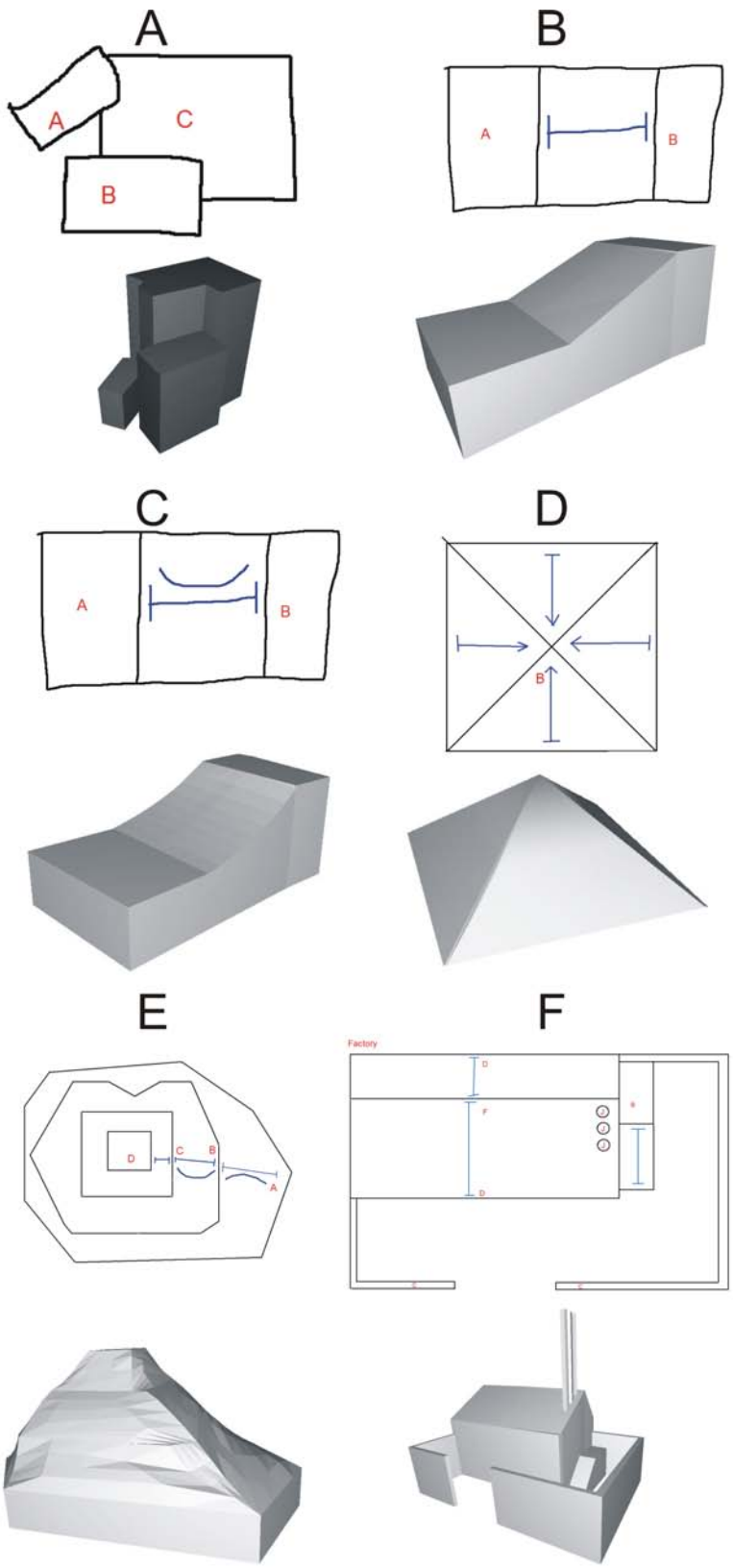
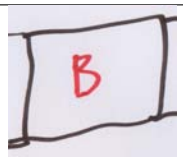
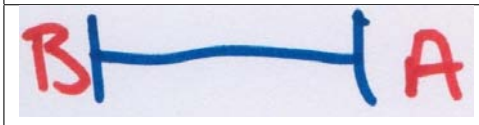
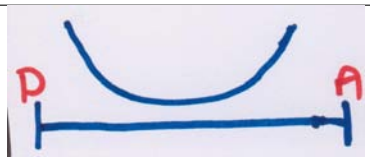


Figure 4.1: Input images and their resulting meshes.

Table 4.1: Graphical table of language features

Feature	Description
	Height marker
	Closed region
	Interpolation line
	Interpolation line with a concave curve descriptor
	Spider

produces three volumes with uniform height across their surfaces.

4.1.1 The need for colour

This would be an appropriate point to comment on the choice of a separate ink for text (and semantics). Red ink is reserved in the system for text for two purposes. The first is that by mandated colour separation, visual clutter and feature density is alleviated. The second and more important purpose is to aid the process of text recognition by essentially reducing it to the colour segmentation problem. While there are algorithms for the purpose of separating text from mixed text/graphic environments [22] [55], their simple robustness somewhat relies on the consistent size of individual characters, and makes the assumption that there is little or no connection between the letters in a word. These assumptions are reasonable for printed documents, but for lettering created by hand they seemed unsteady. Because of this the time penalty of developing and refining of a more intelligent text-detection was decided to be unnecessary given the system's prototypical nature, and that a mandated ink colour for text was a reasonable constraint.

4.1.2 Spiders

Height values can be also assigned at a distance with use of pointers in the semantic colour layer. This is illustrated in Fig. 4.2. Such a pointer (called a spider for the reason that it looks like one) structure moves and duplicates whatever text is

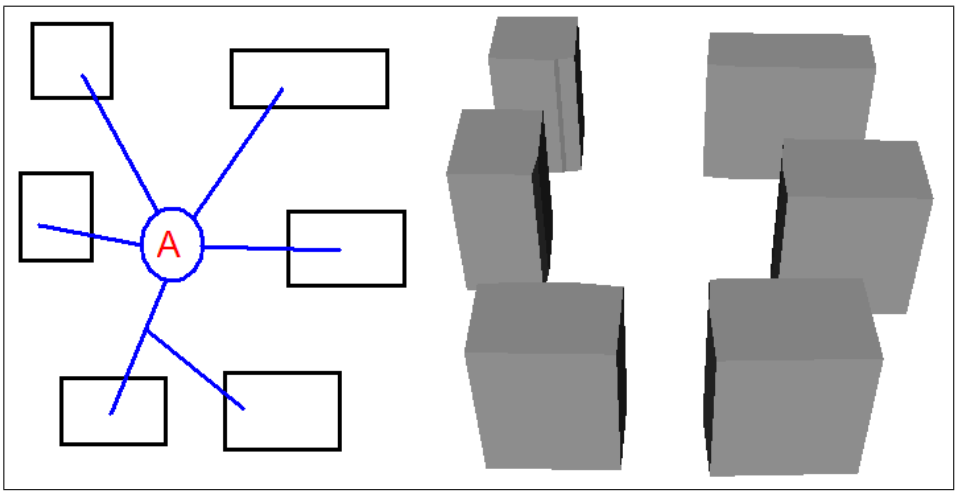


Figure 4.2: Height markers assigned with a ‘spider’.

in its ‘body’ and places one copy at each of its ‘feet’. This is useful for labelling regions with text that may not fit in them, or for quickly labelling multiple regions with the same height.

Spiders are interpreted before any semantic meaning is extracted from text, so they also work with map references, described in §4.4.

4.2 Interpolation lines

To provide sloped surfaces, the language introduces a semantic construct called an interpolation line. Such a line is used to express a height function that interpolates its values between two points. Fig. 4.1.B gives a simple example of such a line in blue ink. The closed region it occupies requires no height marker, as the line indicates that the region will derive its height from the two adjacent regions it

points to.

Fig. 4.1.D illustrates the variation in arrow-heads that can be attached to interpolation lines. The lines in D, with sharp arrowheads at one end and flat heads at the other, indicate that these height interpolations are to be between a point on the respective closed region's boundary and a flat segment.

4.3 Interpolation curves

The addition of interpolation lines is enough to make the language a geometrically (if not semantically) expressive super-set of standard isoline-based topographic maps, since closed regions provide for the description of vertical walls, which pure contour lines cannot. Where the language loses out to topomaps is in a comparison of ink usage. Not only does the language have to provide contour lines and height references, but it also requires that each closed region include an interpolation line.

The addition of another language feature improves this trade-off. We allow the user to draw a curve alongside interpolation lines, to indicate that the height interpolation it describes is not to be linear (and thus producing a ramped surface), but curved. This is illustrated in Fig. 4.1C showing a concave curve and two curved interpolation lines in 4.1E.

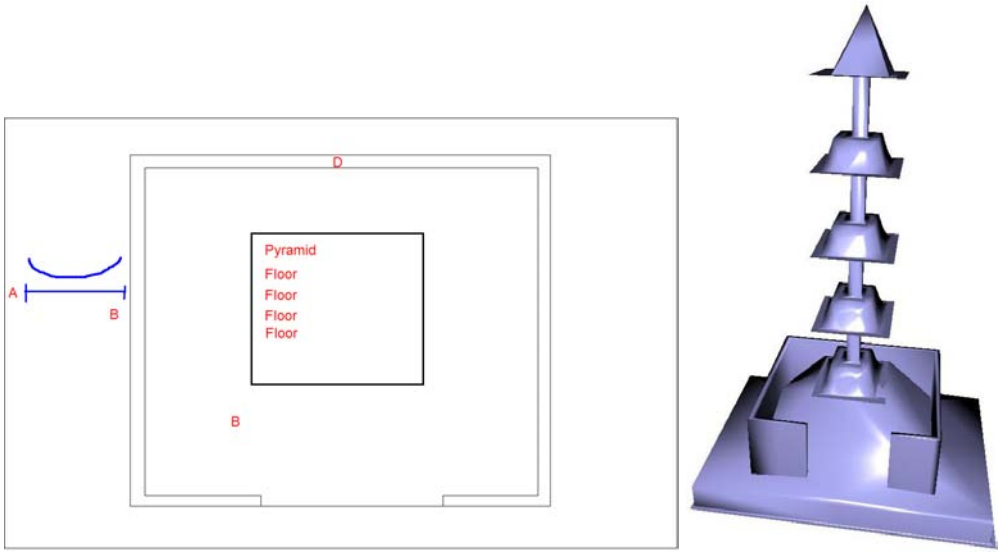


Figure 4.3: A mesh composed out of a stack of other meshes referenced by their names. The embedded meshes appear vertically in the order they are listed vertically in the inner closed region of the input map.

4.4 Naming, referencing and composing models

Several usability concerns and desirable attributes are addressed by supporting map naming and referencing. The first usability concern this addresses is that working with a finite sheet of paper at a fixed scale imposes severe restrictions on the geometric 'resolution' at which the user can work. If we allow for individual maps to be uniquely named, and then referenced by name in other maps, this scaling problem can be solved.

This feature has other benefits. A common characteristic of artificial landscapes (i.e, buildings and other structures) is a reuse of geometric elements such as spires and pillars. Combined with spidered labelling, duplicating a mesh sev-

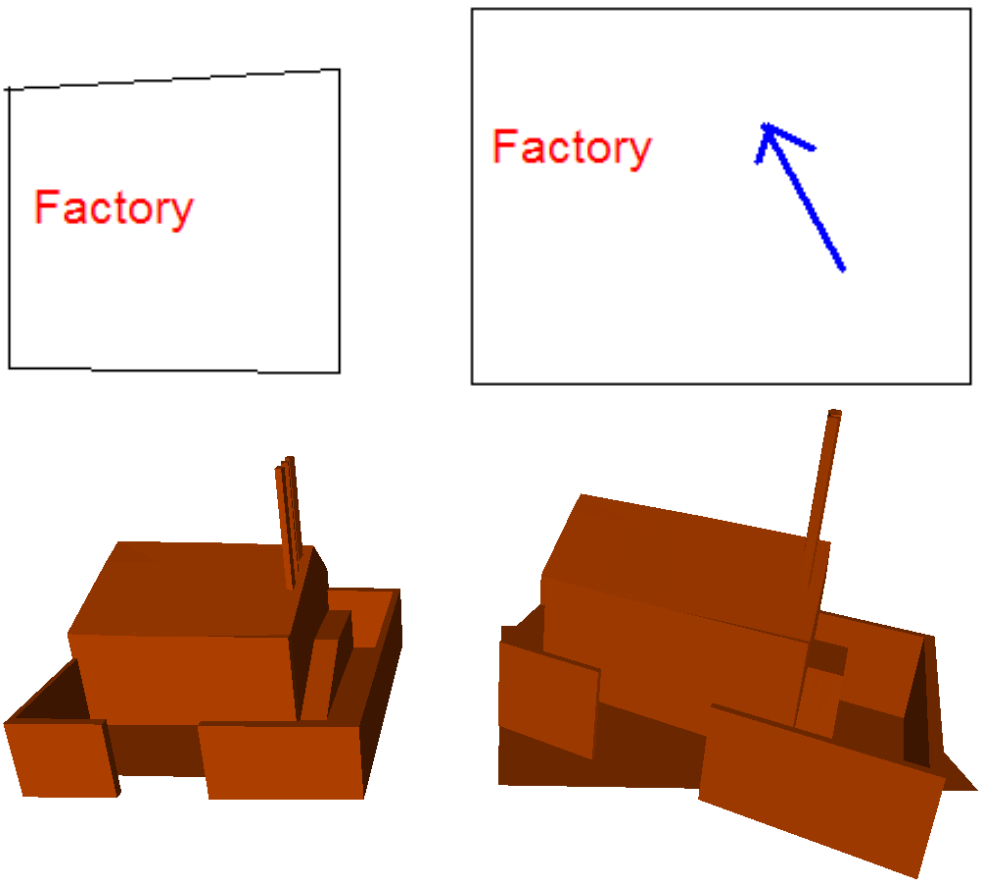


Figure 4.4: Two referenced maps, one rotated with a directional arrow

eral times is made possible. Additionally, as is often desirable in buildings and building-like geometry, meshes can be stacked upon each other as in Fig. 4.3.

4.4.1 Rotation referenced models

The orientation of referenced maps can also be changed with use of a blue arrow. If one is found in a referenced map region, the referenced map is rotated so its "north" is aligned with the arrow. Without an arrow, no rotation is performed,

which is equivalent to the presence of a vertical arrow.

Chapter V

System structure and implementation

The system is structured as a non-interactive pipeline, receiving one or more digital images as input. This chapter describes that pipeline and its steps in detail, as well as the decisions and problems that arose at each step. The pipeline is diagrammed in abstract in 5.5

5.1 Pre-processing

Depending on the origin of the input image, some amount of pre-processing may be required to make it suitable for the pipeline's consumption, a task which can be generally described as noise removal, where noise is any undesired feature in or of the image that will hamper the pipeline's process or accuracy. Noise can come from multiple sources, particularly if any sort of real world input is involved, such as a scanner. Obstacles in that case can include paper that is dirty or not lying flat on the scanner bed, ink that is smudged or fading in parts, or thin paper with ink from the other side showing through the surface. Some of these like the colour bleeding in Fig. 5.1 can occur systematically throughout an image. Chromatic aberration is an optical phenomenon associated with different wavelengths

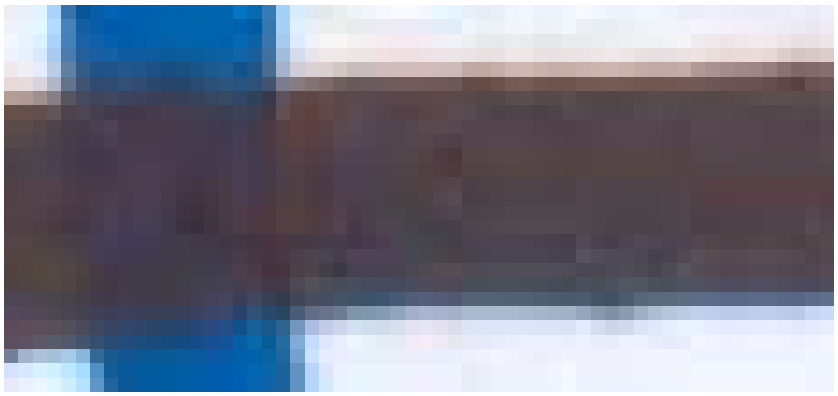


Figure 5.1: Red/blue chromatic aberration surrounding an area of ‘black’ ink.

of light having different refractive indices as they pass through the scanner’s lens, and manifests itself around any areas of high contrast, such as where black ink meets the surrounding white of the paper. It also manifests in high-resolution images taken by consumer grade cameras, and its handling or removal in both post-processing and at the time of image capture is an active research area[56].

Systematic red-blue aberrations like the type in Fig. 5.1 can be particularly problematic for this system, as the potential exists for the coloured fringes that surround marks to be segmented as separate entities into their respective colour layers, creating large artifacts that size thresholds (i.e, a filter that removes all discrete components below a certain size) will not eliminate.

As Fig. 5.1 also illustrates though, at the pixel level there can be quite significant variation between the ideal solid black and the actual pixel values found in scanned black ink. This, the chromatic aberration, and any other source of noise are addressed via a series of steps in the early steps of the system, before and after

colour segmentation splits the input image into its semantic layers.

5.1.1 HSV colour contrasting

The first concrete step in preprocessing is a contrast enhancement operating on the saturation and value dimensions in the HSV colour-space. A typical contrast enhancement function, given a midpoint m and enhancement ratio r where $0 \leq m, r \leq 1$, will operate in RGB colour-space on either a per-channel or averaged-channel basis, on each pixel with intensity I such that its new contrasted value I_C is

$$I_C = Min(1, Max(0, \tfrac{I}{1-r} - \tfrac{1}{2-2r} - m + 1))$$

which is plotted in Fig. 5.2. It is a simple linear equation constrained to fall within the 0–1 range. Other mappings such as sigmoidal functions are also useful for this role. [57]

Run multiple times or with a sufficiently large r value, the final state of such an operation is a binary black and white image (assuming the contrast is being performed on all three colour channels), which by itself is unhelpful. A more useful approach is to use the HSV colour space, illustrated in Fig. 5.3. By performing a contrast operation on the saturation and value channels, the colours in the image are driven towards the preferable and expected areas in the colour space: white, black, and vivid colours (such red and blue). Doing this prior to the colour segmentation stage with an appropriate adjustment ratio r (0.2 has proven

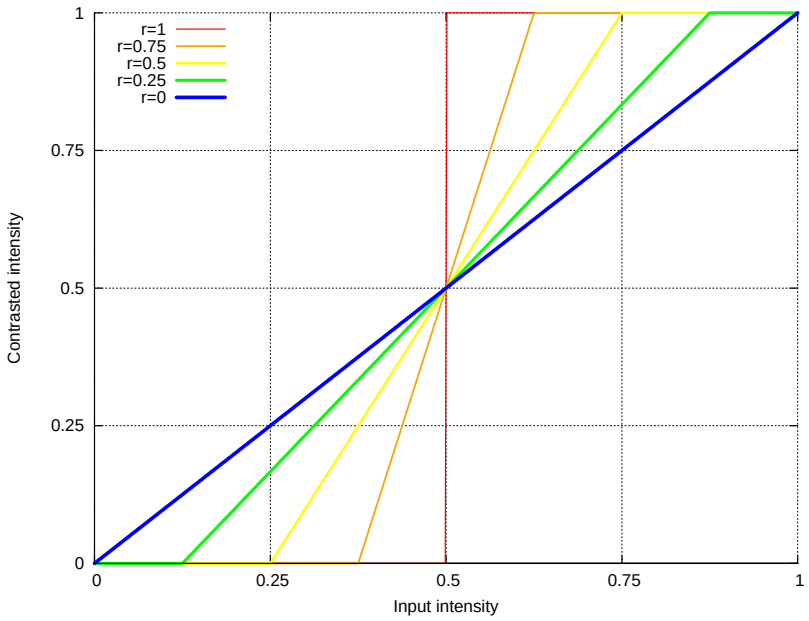


Figure 5.2: The contrast function with $m = 0.5$

to be suitable for scanned images) improves segmentation results in two ways. By 'burning' and 'blacking' a significant number of pixels (i.e, pushing them to the edges of the colour distribution to become pure white or black), the discrete number of unique colour values in the image is reduced. This is most useful in eliminating any faint tinge or hue that the scanning process has left in the white paper areas, as the white or near-white pixels tend to otherwise dominate an image, and the *k*-means algorithm described in the next section does not perform well when there is a large difference in the size of the clusters it separates. The practical consequences of this can be seen in Fig. 5.4. Additionally, increasing the contrast in this way increases the distance between the colour-space clusters, aiding the *k*-means algorithm by decreasing the ambiguity of which cluster a colour

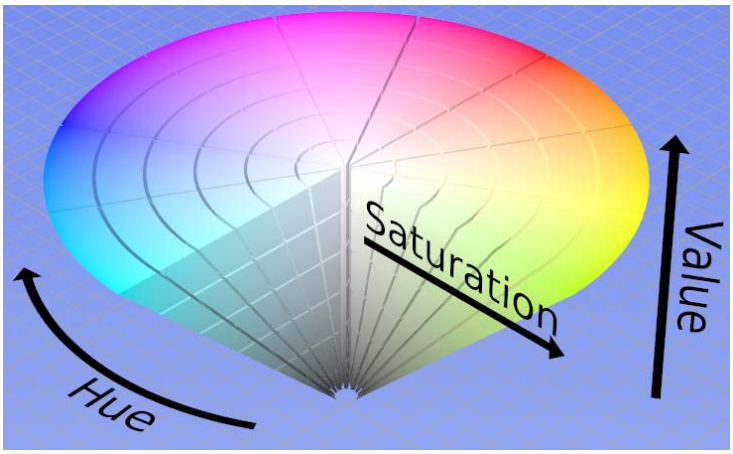
cone.png 

Figure 5.3: Conical depiction of HSV. Image from [58]

belongs to.

Currently the system operates on the assumption that the input image is either from a flatbed scanner or directly drawn via a tablet device. If it was to be extended to support other sources such as a whiteboard, this would be the step in the pipeline for such a subsystem as described in [59] that handles the requisite tasks such as image stitching, whiteboard identification, and perspective correction.

5.2 Colour segmentation

The first stage (after any pre-processing) segments the image into up to three binary images of the same dimensions by colour. The segmentation is performed in *RGB* space with *k*-means[60] using a slightly modified version of Lloyd's algorithm[61], a suitable and reliable approach since we have a-priori knowledge of what colours the images are likely to be drawn in. The algorithm's clusters

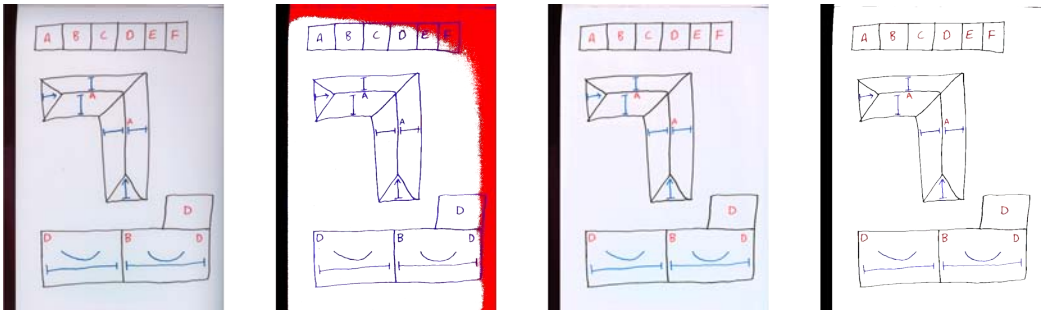


Figure 5.4: The impact of an HSV contrast operation on colour segmentation. FAR LEFT: The original image. INNER LEFT: The result of its segmentation into black, white, red and blue. INNER RIGHT: The original image after HSV contrast with $r = 0.4$. FAR RIGHT: The result of segmentation after HSV contrast.

are pre-seeded to reflect the colours expected to be in the image. The pre-seeding proves to be a critical step for input images taken from real-world sources such as a flatbed scanner. This is illustrated in Fig. 5.4, where a colour distortion resulting from the physical characteristics of the scanning process gives the resulting image a colour tinge that fades from red to blue across the entire image. As this most noticeably affects the blank (i.e., white) portions of the image paper, which makes up the majority of all the pixels in the image, a segmentation based on global colour distribution (such as *k*-means) is vulnerable to image-wide noise.

k-means normally requires *a priori* knowledge of k (how many clusters are present), and though there is such knowledge of what the clusters are likely to be for a given k, there is no guarantee that an input image will contain interpolation lines or other blue-ink features, so the algorithm needs to be able to accept images with $k = (3,4)$. Or if the system is to be used in an semi-interactive way, such

as by taking input from a live camera focused on a whiteboard or paper, then it would not be unreasonable to expect input images consisting of only one or two colours (white and black).

Lloyd's algorithm takes a basic iterative approach. When given a population it picks k initial seed values (typically either randomly generated or selected from the population). It then uses these seeds as centroids, clustering the population by some distance function (this implementation uses Euclidean distance in the RGB space) and matching each sample in the population to its closest seed value. The centroid of each cluster is then recalculated as the mean of all the members of that cluster. The algorithm iterates in this fashion, recalculating cluster membership and centroids until it reaches the point of stability. For performance reasons, with an input image above a certain size, this clustered population will be a subset of all the pixels in an image, or based off a smaller version of the image.

Lloyd's algorithm is accordingly slightly modified as follows: Since we can impose an order of necessity in which pixel colour will appear (white, black, red, blue), and ensure that the clusters' seed values align with these colours, we are able to cull any clusters that receive no pixel matches in the first pass of the algorithm. The resulting layers are matched to their original seeds (and ergo semantic intentions), and the white layer (i.e., the background colour of the paper or image) is discarded. The resulting image layers are binary images.

5.3 Geometric layer

Most cartographic maps have multiple layers of meaning composed into a single image. Iconic representations, text labels, topographic lines, roads and other forms will appear adjacent and overlaid on each other, as the cartographer has determined fits best. This system has three layers of meaning present, called the text layer, the semantic layer, and the geometric layer. The latter is in function the layer most closely related to the topographic lines used in conventional mapping. It describes geometry as it will result in the final model: a shape in the input diagram's geometric layer will appear with its planimetric form preserved.

5.3.1 Growing and shrinking

The geometric layer (drawn in black) is grown and shrunk to remove any one-pixel holes that may still be present. Growing and shrinking are the process of applying a maximum and minimum (respectively) neighbouring value function to each pixel with a 3×3 filter kernel. This are the morphological operations also known as dilation and erosion. For pixel components above a certain size, this has little aggregate impact on shape or size, but it does fill in single pixel holes that can otherwise be troublesome in the thinning process, as illustrated in Fig. 5.6.

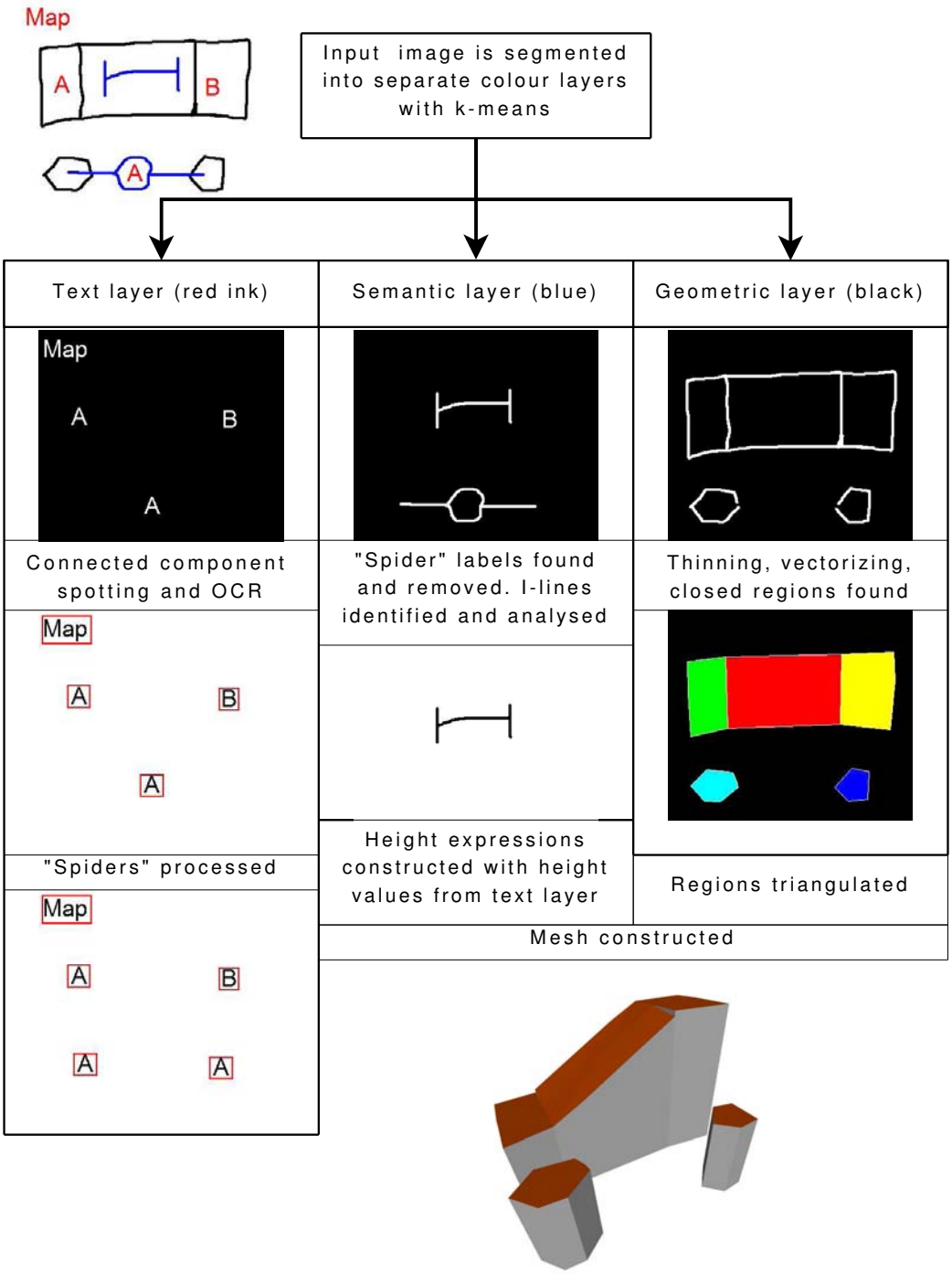


Figure 5.5: System pipeline structure

Figure 5.6: Growing and shrinking an image. From left to right: the input image, after growing, after shrinking. White is the "active" colour.

5.3.2 Binary thinning

The layer image is then thinned. Thinning is a process used to derive the skeleton of a binary image, where the skeleton is the subset of black pixels (assuming black is the 'active' colour) such that the geometry of image features are preserved, particularly connectedness. Ideally the skeleton is only one pixel in thickness. A typical approach is to iteratively erode all features, repeatedly progressively scanning the image for pixels that can be safely removed without breaking local connectivity between features. The system uses Neusius' non-iterative thinning algorithm[62], since its time performance scales well with large input images compared to iterative algorithms that use heuristics to selectively erode pixels away from pixel components until a skeleton is left. The result of a thinning operation is illustrated by the first transformation in Fig. 5.7.

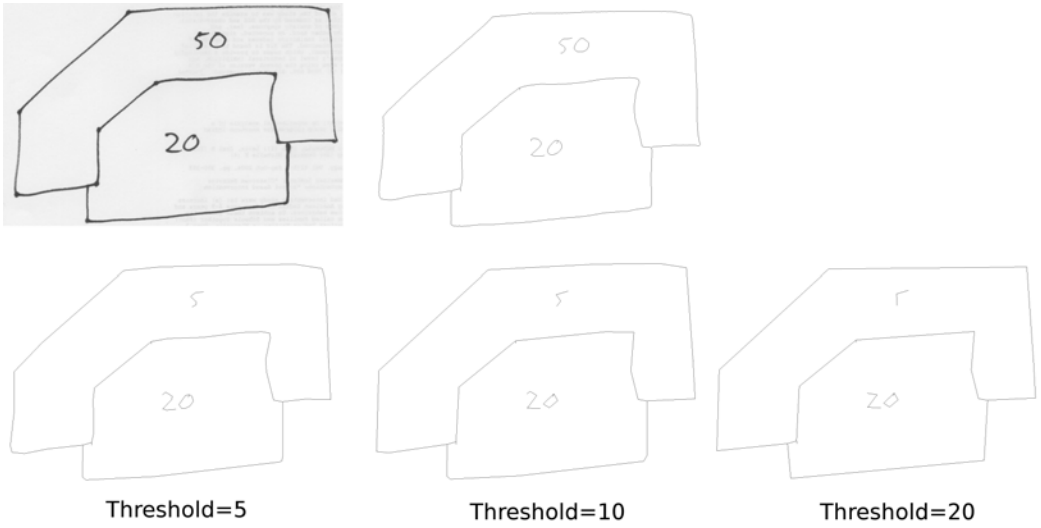


Figure 5.7: An example of the vectorization process. Top left: original input image. Top centre: the image after segmentation and thinning. Bottom: The thinned image after polyline simplification with three different pixel-distance thresholds.

5.3.3 Graph simplification as vectorization

If thinning is properly achieved, the skeleton can be regarded as a graph of connected segments forming poly-lines, where every pair of adjacent black pixels form two nodes and an implicit edge joining them. With this abstraction, the robust Douglas-Peucker (DP) algorithm[63] can provide a means for vectorization. There are problems and intricacies with this abstraction, discussed later in §5.3.4.

The DP algorithm (illustrated in Fig. 5.8) considers a chain of points describing a line, constructs a line c between the first and last points, and finds the point m on the polyline furthest from c. If the distance does not exceed the threshold, c is deemed a good simplification of the chain of points. If the distance does exceed the threshold, the algorithm replaces c with two more constructed lines, one from

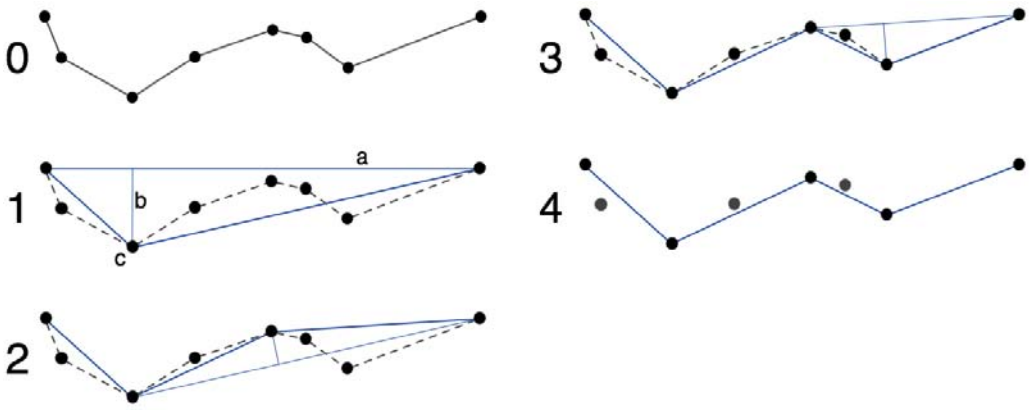


Figure 5.8: The Douglas-Peucker algorithm. Image from [64]

the start point to *m*, and the other from *m* to the end point. It then recursively applies itself to these lines to test if they are appropriate simplifications. Fig. 5.7 illustrates the results of different threshold distances applied to the same skeleton. The pipeline provides the algorithm with all the pixels in the thinned lines as initial input.

It should be noted that like many steps in the pipeline, this is not a particularly sophisticated or domain-specialised variant of the algorithm, and there is doubtless opportunity to improve on output quality. E.g, in using a measure of furthest distance from the new line as its heuristic for whether to make a simplification, classic DP takes no other local or neighbouring factors into account. As a result, deliberately curved features such as circles vectorize coarsely in the current system, and edges that the user to be obviously straight or parallel are vulnerable to noise that domain specialization of the algorithm could improve upon. The final step in vectorization is a reconnection process with a small distance threshold

(typically 10 pixels) between endpoints. This provides some robustness in the face of lines being severed by intersecting lines.

5.3.4 Closed region identification

After vectorization, the resulting set of poly-lines is treated as a planar spatial directed graph G where $E(G)$ is the set of segments in the image and $V(G)$ is the set of vertices to which they belong. A complementary pair of edges is made for each segment.

From G we find all closed regions with a series of walks. We declare the set

$$J = \{v \in V(G) \,|\, d(v) \in \{1, 3 \ldots\}\}$$

where $d(v)$ is the valence of a vertex. J forms the set of vertices that act as the end-points of the poly-lines made out of the segments in $E(G)$.

We find the closed regions (equivalent to faces in the domain of planar graphs) by making progressive walks through G. At each vertex that is in J, the next most counter-clockwise (CCW) vertex is selected to visit. If a vertex has already been visited in that walk, the walk's edges are removed from the graph. Carrying this through until $|E(G)| \leq 1$ gives us $r + g$ walks, where each walk r is a CCW description of a closed region, and each walk g is a clockwise (CW) description of the perimeter of a cluster of vertex-adjacent closed regions. These perimeter vertex lists are useful further on in the pipeline (particularly triangulation) when

dealing with region nesting, as they define the holes that need to be "cut" into the surface of an enclosing region.

Handling degenerate output graphs

This simple approach is not without its intricacies which quickly befall a naïve implementation. The abstraction of an 8-connected binary bitmap image into a graph of edges connecting nodes will present difficult geometry if performed absolutely rigourously. An example like Fig. 5.9 illustrates this. Many thinning algorithms will render intersections like 1 and 2 in Fig. 5.9 where, although the pixels marked *X* are not necessary to maintaining 8-connectedness, the geometry's straight edge is better preserved if it is left intact.

But a transfer of an interection like that in Fig. 5.9.1 to a graph results in the creation of two spurius cycles at the point of intersection. Left untreated, these cycles will be treated as closed regions.

Even if a thinning algorithm enforces minimal 8-connectedness (i.e, no pixel is left in the skeleton if its removal would not affect the connectivity of its neighbouring pixels) as it is in the third intersection, it can still produce a spurious loop in a 4-way intersection, as the fourth illustrates.

Finally, there is no guarantee that the noise removal in the previous steps will have been totally effective, and so some small loop may otherwise survive even if none resulted from the graph creation process.

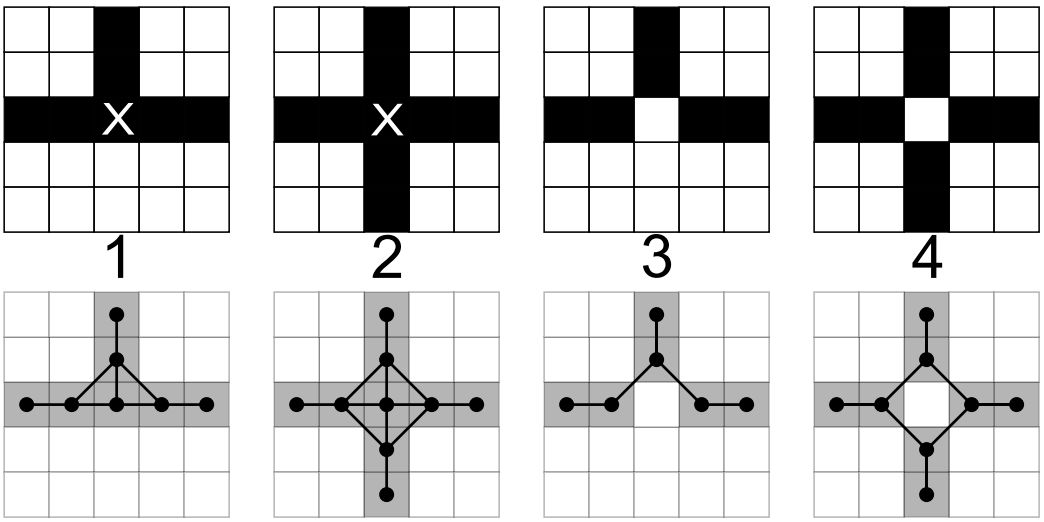


Figure 5.9: Consequences of thinning policy on resulting graphs. TOP: Four examples of two types of intersection. 1 and 2 are produced under a lenient (i.e, geometry preserving) thinning policy, 3 and 4 are produced under a strict (i.e, 8-connectedness preserving) policy. BOTTOM: The resulting edge–node graphs.

This necessitates another simplification of the graph. The system identifies all clusters of joint vertices (those connected to either one edge or more than three) that are themselves adjacent to each other, and replace each cluster of joints with a single joint in the average position of the joints.

5.4 Text processing

The text layer (drawn in red ink) is split into its binary pixel components. A binary pixel component is a collection of 8-connected pixels. The components are identified using a single-pass component labelling algorithm similar to the typical row-iterative one described in [65].

The components are then clustered according to pre-set distance thresholds, and the clusters rendered into individual images, ready for passing to external OCR[1] programs. Currently we use Tesseract[66] with GOCR[67] as a fallback in case of recognition failure. From this we are given a set of text labels, and retain the centroid of the original pixel component clusters to regard as the text labels' respective positions. The labels are first sorted by their distance to the upper-left corner of the image, and if the closest label is within a certain threshold distance, it is set aside as being the name of the map. This is used later to give the finished mesh object a filename, which in turn can then be referenced in another map.

The OCR programs employed by the system are primarily designed for recognition of printed characters of a uniform upright orientation, laid out in horizontal lines. The system ameloriates this latter expectation by passing in each label separately after performing its own component identification, but it does pass onto the user the requirement that text be of a uniform and upright orientation. Additionally, an OCR library trained on printed fonts can be expected to suffer when tasked with handwritten text, and even fairly clear handwriting can present problems.

The problem domain allows for guidance of the OCR process to improve its accuracy. Any text considered by the system can be one of three types: a name for the map in question, a height marker, or a referenced map. All three have practical and helpful constraints. A map's name must occur within a certain distance of

[1] Optical character recognition

its originating image's top-left corner. A height marker must be from either a very limited subset of possibilities (in the case of an alphabet code) or at least compromised from a limited set of characters (in the case of numeric markers).

5.5 Semantic layer

The final layer to be described is the semantic layer. Its purpose in the system is to provide an area for the abstract symbols that manipulate the shapes in the geometric layer.

Like the geometric layer, the semantic layer (drawn in blue), once received as a binary image, is grown and shrunk, thinned, vectorized, and converted into a set of polylines. The connected (or nearly connected, according to some distance threshold) poly-lines are aggregated into discrete entities, which are then checked by a series of heuristic pattern matchers.

5.5.1 Spiders

The first is the test for whether an entity is a spider as described in §4.1.2. This is done by checking for the presence for an inner loop, using the same algorithm for finding closed regions in §5.3. If one or more closed regions are found of a sufficient size (A noise avoidance measure. A low threshold is sufficient.) that contains a label found in the text layer, the largest region is treated as the spider's 'body'. All remaining trailing vertices in the entity (i.e, every vertex that only

connects to one other) are treated as the spider's 'feet'. Then in the data structure holding the text labels, the original label is removed, and duplicates placed at each of the feet. Because this transformer applies to generic text labels regardless of its semantic value it is useful for both ascribing height markers and mesh embeddings, both in areas where close duplication is wanted, and as a way of attaching text to regions too small to comfortably contain the text.

5.5.2 Interpolation lines

Entities which are not recognised as spiders are then tested as being interpolation lines. The first stage of this is to find the longest polyline in the entity (remembering that polylines are regarded as stopping at ending at vertices that connect to vertices with a valency other than two), and regard this as the body of the line. The poly-lines at each end of the line are analysed to determine their type. This is made simple in the current system as there are only two types of arrow-head supported (other than lines that terminate without an arrow-head); flat-heads for indicating a segment and arrows for indicating a point or corner. This is done simply by constructing a rectangle around the vertices in each arrow-head, axis-aligned with the direction of the arrow's body, and the rectangle's length to width ratio. If it is less than some constant (0.3 in this implementation) the pointer is decided to be an arrow, otherwise a flat end.

Once the pipeline has progressed to the state that the closed region nesting

hierarchy has been established, all sets of interpolation lines that share a region are examined for the possibility that they are actually an interpolation line and its curve descriptor (described in §4.3). If the closest distance between two identified interpolation lines is below a provided threshold, then the longer of the two lines (as determined by straight endpoint-to-endpoint distance in the line's body) is assumed to be the interpolation line, and the shorter its curve descriptor.

5.5.3 Curve descriptors

From the curve descriptor we wish to find $-1 \leq c \leq 1$ where c is a description of the convexity of the curve over the resulting height descriptor: we want -1 to correspond to a maximally concave curve, 0 to correspond to a linear interpolation, and 1 to correspond to a maximally convex curve. We first find the vertex d_{max} on the curve descriptor poly-line $d_{0\ldots n}$ that lies the farthest from the segment (d_0, d_n) with distance w. From this we find

$$c = Max(Min(\frac{wks}{Distance(d_0, d_n)}, 1.0), -1.0)$$

Where k is a constant established by preference (2.5 in this implementation), and

$$s = 1 \text{ if } (\frac{\sum_{i=1}^{n} x_i = Distance(d_i, A)}{n}) \geq 0 \text{ else } -1$$

where A is the body line segment of the interpolation line proper. So the

weighting c of the curve descriptor is established by a comparison of the descriptor's own maximum distance from the segment formed by its endpoints, with the length of that segment. Whether this describes a concave or convex curve is then decided by whether the vertex at which the maximum was found is closer or further from the end-to-end points of the interpolation line's body than the average distance of vertices in the curve descriptor.

5.6 Height expressions

Having identified all the relevant discrete features in the image, the system then attempts to build a height expression for each closed region. A height expression E is a function that takes a 2D planar co-ordinate p and returns the respective z value.

The height expression for an entire region can be composed of multiple individual height descriptors. When there are n descriptors (E) in a region, the system creates a blended function E_B as its over-arching height expression. d_i and d_{total} refer to the individual distances and the sum of the distances between the height expressions (or rather, their originating entity) and p:

$$\mathrm{E}_B(p) = \sum_{i=1}^{n} \mathrm{E}_i(p) \frac{d_i}{d_{total}}$$

The nature of these separate descriptors is varied. In the case of a text label that

stands by itself (or is beyond any applicable threshold distance to another feature like an interpolation line), and is found to correspond to a height value in the global look-up table, it will return a constant value matching that value's table entry. In the current system, the height entries match the first letters of the alphabet: $A = 100$,$B = 200$,$C = 300$, etc. Fig. 4.1A illustrates three closed regions, each with only one height marker. Each of these regions in turn only produce a single height expression, resulting in flat topologies when meshes are constructed.

The complexity in the height expression system comes with interpolation lines, which require a series of different behaviours dependent on their own and their neighbouring regions' contexts.

In the local context, the interpolation line may have height markers in the same region close by it to assign height values to either end, as seen in Fig. 4.1 D, E, and F. Alternatively one or both ends of the interpolation line may be unadorned, particularly illustrated in 4.1 B and C, requiring the height expression for that region to 'consult' its neighbours and see whether they are able to provide height information for the target segments. Given that these neighbouring regions may have their own interpolation line expressions to resolve (or that may not be possible to solve yet with the known information), this can become complex quickly. At present the system does not try to resolve such information beyond the first degree of neighbouring regions.

Additionally, the structure of the region and its neighbours can have an in-

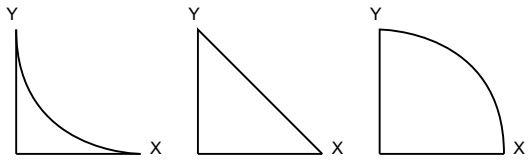


Figure 5.10: The curve attribute height equation, with c values -1, 0, and 1.

fluence. For each of the ends of an interpolation line, the system tries to solve a height and target vertex set. For each end, a ray is casted along the direction of the interpolation line until an intersection occurs on the geometry layer. If the ending type is a flat arrowhead, the target is taken to be the intersecting segment. Otherwise it is taken to be the closest vertex on that segment to the intersection point. The exception to this is found when one of the targets is part of a nested sub-region of the current region and the other is not. In this case the target vertex sets are taken to be the entire boundary of the sub-region and the parent region, creating standard isoline-based topographic map behaviour.

In the absence of curve descriptors, the interpolations are assumed to be linear.

5.6.1 Curved interpolations

If a curve descriptor is associated with an interpolation line, with a curve weight $-1 \leq c \leq 1$ (as calculated in §5.5.3), the height value range and horizontal range are normalized to fall between 0 and 1. The curvature used is the arc of a circle: at $c = -1, 1$ the curve is an arc of 90°, at $c = 0$ the arc's size is infinitesimal, and treated as a straight line, as illustrated in Fig. 5.10.

The radius and center of the circle are calculated so that it intersects the positive x and y axises at 1. So for curve weight w

$$d = \frac{1}{w} - 1$$

where d is the displacement of the circle's center from the origin point $(0,0)$. To maintain the correct intersection with the x and y axes, the radius r is calculated as

$$r = \sqrt{2d^2 + 2d + 1}$$

This results in the height equation

$$y = \sqrt{r^2 - (x+d)^2} - d$$

where x is the normalized horizontal value between the two ends of the interpolation. If $c < 0$ then $1 - x$ is used as the input instead, to invert the curve. The height curve is stretched to fit the horizontal and vertical bounds of the interpolation.

5.7 Mesh construction

At this point the pipeline has finished the task of interpreting the input image and is now focused on building the interpreted structure into a 3D mesh. The

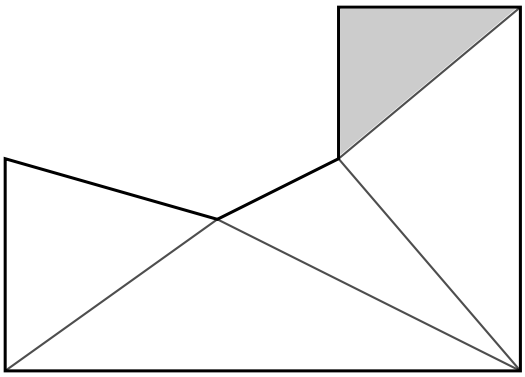


Figure 5.11: An ear of a polygon

closed regions are converted to a planar set of 2D triangles first by a simple "ear-clipping" algorithm, similar to those described in [68]. Ear-clipping is the process of iteratively finding and removing as a triangle an "ear" on a simple polygon. An ear (Fig. 5.11 is an ordered triple of successive vertices in the polygon's perimeter, where a segment can be cast between the first and last vertices that will fall inside the polygon without intersecting any part of the perimeter.

Ear-clipping requires a simple polygon (i.e., one without any holes), so a pre-triangulation phase of cutting edges from the outer boundary to the inner holes that surround inner nested regions is used. This approach for extending ear-clipping is discussed in [69]. Note that these inner holes are not necessarily the same as the boundaries that surround the individual sub-regions, but are the boundaries of each joined cluster of sub-regions, as generated by the poly-line graph walk described in §5.3.

For regions with non-flat surfaces, triangles are subdivided several times ac-

cording to size to make for smoother curves and surfaces. Then, for each triangle in each region, the region's height expression is used to convert the triangle into 3D with the appropriate z-values. Regions that immediately adjoin each other with abrupt height changes in the surface (i.e, neighbours with height differences that will produce vertical walls) are identified and the vertical skirting triangles to create the vertical surfaces are added appropriately.

To maintain some visual distinction in the output, triangles created from closed regions and triangles created from the skirting process (i.e., vertical surfaces) are flagged for rendering in different colours in the output mesh. Objects are saved in the Alias Wavefront OBJ format [70].

5.8 Referenced mesh embedding

If a name for the map is found as described in §5.4, the model is saved with that as a filename. Once the system has produced a 3D mesh, it examines all text labels that have not been recognised and treated as height markers. It compares these labels against a list of filenames of available mesh files, and matches pairs between the two lists, using the Levenshtein edit distance [71] as a comparison method to allow for some flexibility in handling errors from the OCR process. The system does not distinguish between meshes created by other maps and third party sources.

Having matched such labels to mesh files, the system sequentially loads each

mesh and transforms it to fit to the region the text label is in. This involves scaling the meshes along the x and y axises until their maxima and minima fit within the bounds of the region. If there is a semantic-layer arrow also present in the region with no height markers attached, it will be treated as a z-axis rotation marker, such that an arrow pointing upwards will give no rotation, one pointing to the right will rotate the mesh 90 degrees CW, and so on.

This is an admittedly unsatisfactory approach to region fitting, and there is room for a more sophisticated implementation. Ideally, a shape that is clearly a rotation of the perimeter of the map it internally references should not require an extra annotation marking it to be a rotation. This could possibly be solved by extending the text recognition system to handle text at differing rotations, so that the map reference itself could denote the rotation to apply by its own orientation.

5.8.1 Open questions

The question of rotation aside, there is still a problem that comes from the ambiguity of intent that can be read into a closed region with an embedded mesh that does not closely match the shape or height and width of the region: is the mismatch because the user simply has not drawn accurately, or are they intending that the dimensional ratio of the map itself should be altered? And even if is determined to be the former case, the solution is not straightforward. In the case of a square map embedded inside of a non-square rectangular shape, should

the map be expanded to match the shorter or longer axis of the rectangle? Either approach leads to either not all the shape being used, or the map stretching outside the boundaries of the region. This question applies particularly to multiple maps arranged by reference to be adjacent to each other. Without some automatic fitting, there are likely to be either unexpected gaps or overlap between adjacent models in a naïve implementation using either approach.

A related question also applies to the case of referencing multiple instances of the same map. If the same map is referenced multiple times, and the shapes in which they are embedded are relatively similar, should it be taken that all such instances are intended to be the same size? A reasonable solution would seem to be threshold-based: when the same map is referenced within multiple shapes of a similar enough size, the referenced maps would be clamped to the same size.

Finally, if multiple mesh names are found in a region, the system sorts them vertically and loads them from the bottom upwards. At each mesh, a 3D axis-aligned bounding box (AABB) is found, and the following meshes transformed upwards on the z axis so that the next mesh's lower boundary starts at the current mesh's upper boundary.

5.9 System implementation

The system was primarily written in Python as two non-interactive programs, with the initial HSV contrast routine implemented for performance reasons in C++.

The first component (also implemented for speed in C++) took the original input image and performed colour segmentation into two or three separate binary images of the separate colour layers. The bulk of the pipeline resides in the second program.

Performance in this program was vastly and easily improved over a naïve implementation of the pipeline by effective parallelisation of concerns. The program splits off a separate thread to handle each of the colour layers as it requires: the geometric and semantic layers move straight to vectorization, while the text layer identifies each binary component in its layer, and once it has identified neighbouring components as being part of the same string, is able to parallelise the task of removing that component from its layer and passing it as an independent image to the third-party OCR programs. This ease of parallelisation is pleasing in light of the trend that is seeing (and is likely to continue to see) CPUs with increasing core counts.

Chapter VI

Evaluation

6.1 Description

To evaluate the system on its merits as a language that can be understood and used with a reasonable level of practice by a novice, a qualitative user study was conducted.

6.2 Participants

Nine participants (five female, four male, between the ages of 19 and 25) were selected with no particular criteria.

6.3 Method

The study was conducted on an individual basis. Participants were given the reference sheet in Appendix A as an introduction to the language. The tutorial is arranged to introduce each of the major language features incrementally.

After reading each page (which introduced a new concept), participants were

Table 6.1: Tutorial topics and modelling tasks

Page	Topic[s]	Sample tasks
1	Closed regions, height markers	A simple staircase
2	Slopes (interpolation lines)	A pyramid
3	Curved slopes	A skateboard half-pipe
4	Model naming and re-use	Naming a previously made model and re-using it in a certain pattern
5	Spiders	More complicated re-use. A castle

tasked with sketching a diagram on paper that would require the use of the features just introduced. The order of introduction, and sample diagram tasks are listed in Table 6.1. The target shapes were simple and described verbally. To aid the user in working out what height values to use in constructing the model, a cardboard guide was provided (effectively a length of card with incremented markings), printed to scale with the alphabetical height values spaced at regular 2 cm intervals. At the end the participants were asked to fill out the post-study questionnaire in Appendix A, a series of questions on a seven-point Likert scale.

6.4 Results and discussion

6.4.1 Closed regions

The first page of the tutorial (and thus the first task assigned) was introducing the concept of describing a 3D shape as a set of closed planar regions, and using height markers. The assigned shape to draw for this was a staircase, a simple series of

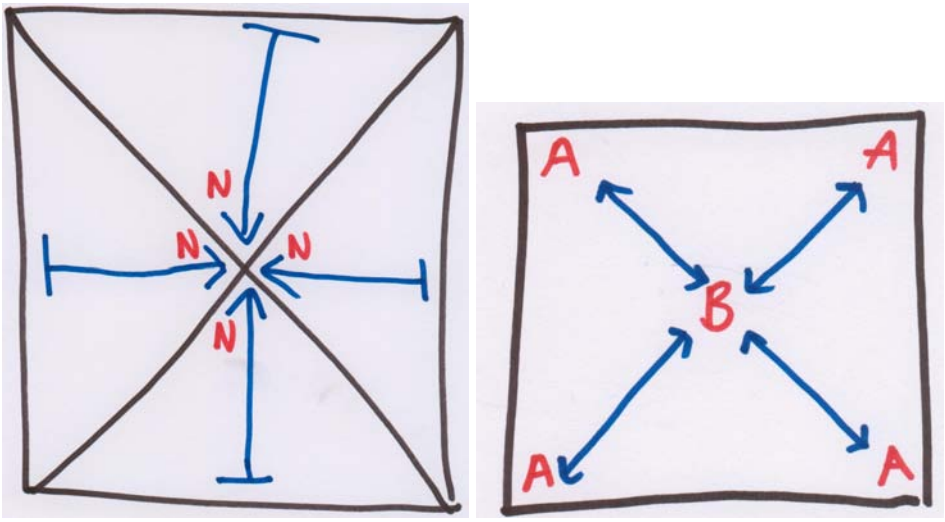


Figure 6.1: Two attempts at drawing a four-sided pyramid.

closed regions, with height markers in ascending value. There were few problems encountered with this task, though being the first in the study some participants made mistakes with remembering to use different ink colours. This was a sporadic problem that reoccurred throughout the test with several users.

Participants found the concept of closed regions relatively easy to grasp: mean 1.5, $\sigma = 0.76$ on a scale of 1 ("Very simple") to 7 ("Very hard"). The task of actually mentally breaking down a desired 3D shape into closed regions proved harder: mean 3.13, $\sigma = 1.36$.

6.4.2 Sloping surfaces

The second page of the tutorial introduced the interpolation line. For this participants were asked to draw a pyramid. Most users drew a diagram similar to

the left-hand image in Fig. 6.1, which would render correctly. The right-hand image would not, but is instructive. The participant said she realised at the time of drawing that it may not work, but it felt like the best way to draw, and this can be understood from the image itself. If the central height marker is regarded as a point itself, or rather as containing a closed region with zero area, then the diagram be correct. This is further discussed in Ch. 7.1.2.

Participants found the concept of making slopes relatively easy: mean 1.63, $\sigma = 0.52$. Like closed regions, they found performing the actual task slightly harder: mean 3.13, $\sigma = 1.36$.

6.4.3 Slope curvature

The third tutorial page introduced curve descriptors. The participants were asked to draw a skateboard half-pipe, an elevated concave curve with a platform at either end. Fig. 6.2 illustrates the mistake that four participants made in this. The left-hand image is the correct diagram, breaking the single curve of the depression into two individual closed regions. The right-hand example will (in the current system) render as a solid with a flat surface, as a height interpolation between two identical height values will be rendered as horizontal, regardless of any associated curve descriptor.

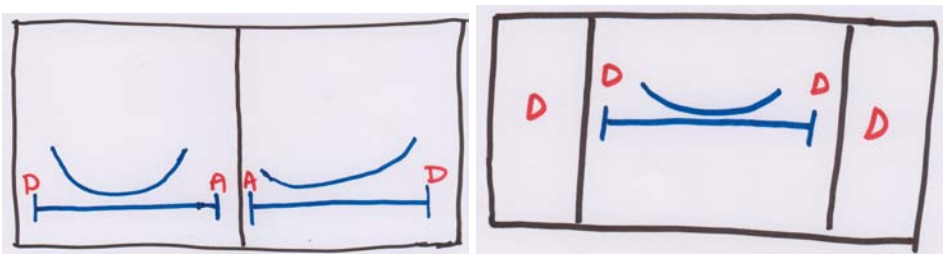


Figure 6.2: Two attempts at drawing a skateboard half-pipe. The image on the left is the correct one.

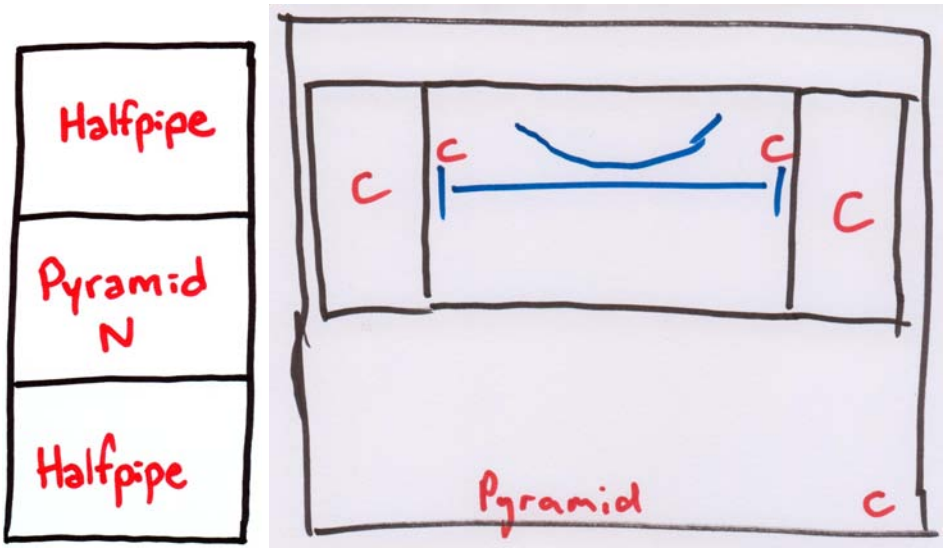


Figure 6.3: Two (incorrect) attempts at stacking a map named “Pyramid” on top of the previously drawn half-pipe.

6.4.4 Naming and referencing maps

The fourth page of the tutorial covered naming and re-using maps. Participants were asked to draw a pyramid stacked on top of the half-pipe they had previously drawn. They were told that the map named "Pyramid" shown in the tutorial sheet was already available to them to use. The intended solution to this task was to name the half-pipe map, and then on a new map stack the two in one closed region. All but one participant correctly drew this, though one drew two version and was undecided between the two. The incorrect version is on the left in Fig. 6.3. In this image, the participant has drawn three closed regions, intending the middle one to be denoted as overlapping the other two by elevating it to a height N, which is the maximum height of the (unshown) half-pipe. The two surrounding regions are intended to be the same region. The participant explained this approach as being a solution to his uncertainty about how to recouncil the differing aspect ratios of the two stacked maps, a problem discussed previously in Ch. 5.8.1.

The right-hand image in Fig. 6.3 is a similar attempt to resolve this uncertainty, except in this version the participant has opted to redraw the half-pipe. The outer closed region is to house the more square-shaped pyramid, with a height marker C intended to elevate it above the half-pipe constructed inside to that height (with the same misapprehension regarding curves illustrated in Fig. 6.2).

This uncertainty emerged in the mean for the post-study question that asked participants to rate the difficulty of "Predicting what size a referenced map would

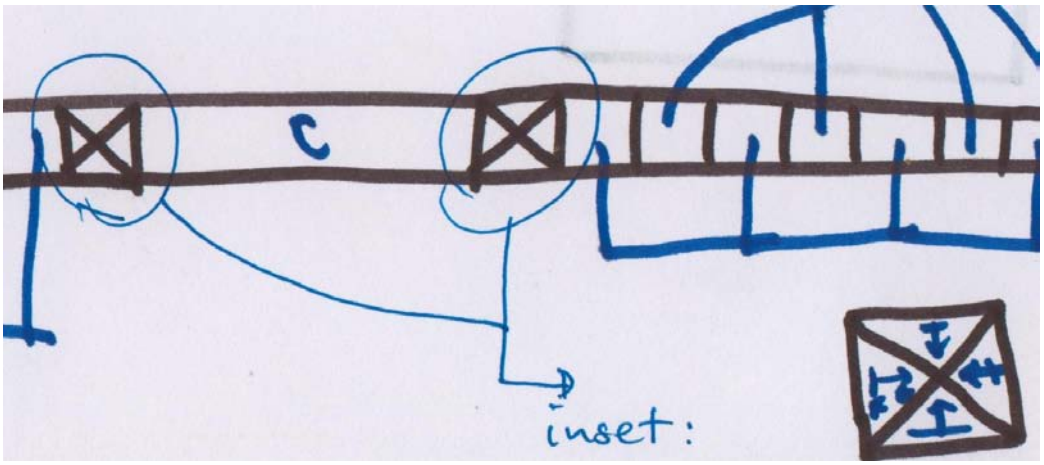


Figure 6.4: A mistaken use of spidering

be", with a mean of 4.17 (the only question to rate a mean above the midpoint of "Average" difficulty, 4) and $\sigma = 1.47$.

6.4.5 Spidering

The final page in the tutorial described spider labelling. The assigned task was to line up a series of pyramids. All participants successfully used spiders for this, with some minor confusion (from two participants) about whether spider legs should enter the closed region they are intended for, or just touch the border.

One participant, however, mistakingly generalized the spider concept into a more useful tool for expansion, illustrated in Fig. 6.4. They used spidering not to insert a piece of text, but a fragment of geometry described beside the main body of geometry in the map. The potential for expanding spidering in this direction is discussed in 7.1.2

6.4.6 General task

The final task given to participants was to draw a castle. This was intentionally vague in its required detail; a target model with room for using all the features described in the language. Fig. 6.5 shows two participants' drawings.

The first makes heavy use of naming and reusing common components, including the merlons[1] along the wall. It uses two spiders to insert a map reference where it would not fit, although to properly align a blue rotation arrow would also be necessary to correctly rotate the referenced "Wall" segment, but this is not a major point of failure, particularly as rotation arrows (Ch. 4.4.1) were not covered in the tutorial. The second castle makes more use of spidering to create its merlons, to the point where the map is dominated by the spider "legs".

[1] The solid raised parts of an embattled parapet or wall

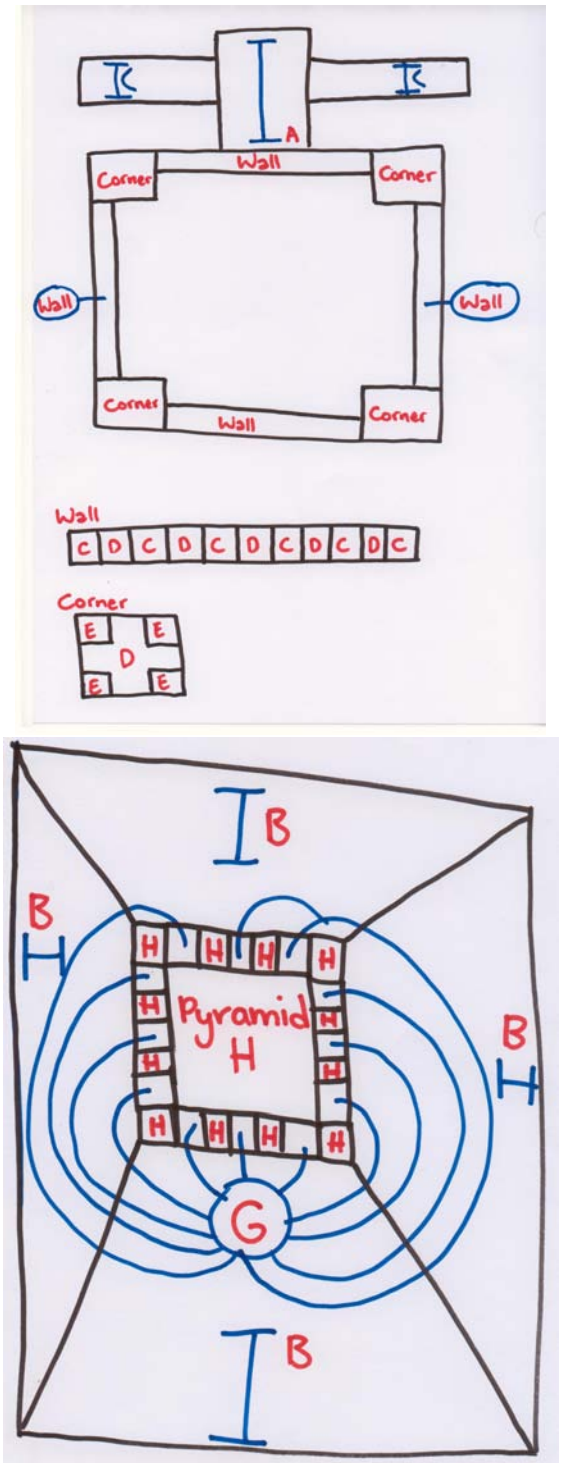


Figure 6.5: Two castles

Chapter VII

Future work and conclusion

7.1 Limitations and future work

7.1.1 Technical attributes

Like most prototypes, there are few points in the pipeline described in Chapter 5 that could not benefit in performance (be it in time or quality of results) from greater algorithmic sophistication. The pre-processing techniques used early in the pipeline, such as the HSV-space contrast and the *k*-means implementation leave much room for improvement. With both of these algorithms, it is worth noting that these particular two techniques operate on a global basis; they treat each pixel in the image independently of its neighbours and local context. Improving on this step would be best started by looking for ways to incorporate the local context of a pixel into the decisions about how to process it.

A similar point can be made about the heuristic post-segmentation noise-removal that operates on pixel components, as all the heuristics currently employed are either intrinsic to the individual component, or extrinsic to the global

population (e.g, removal of components that fall more than a certain distance away from the mean size), but there is room for improving accuracy and robustness with heuristics that also take into account the surrounding context of a pixel component.

7.1.2 Expressiveness and usability

There are many possible ways of increasing the system's expressiveness and usability. Increasing the system's expressiveness could include providing some support for texturing. This would be non-trivial additions to the language's feature set and require some careful integration to fit in well with the design goals laid out in chapter 3.

Many possible usability improvements to the existing set of functionality revolve around a focus on improving performance and reliability in the face of poor input, since the system's primary user interface is the range of input it accepts. There is much room for improvement at all stages of the pipeline, such as at the preprocessing and noise-removal clean-up stage, at colour segmentation, in the binary image operations and vectorization, and so forth.

Peaks and ridges

As the incorrect example in Fig. 6.1 from the user study illustrated, there is room for the language to be expanded in how peaks and ridges are described. This could be implemented by modifying the geometric layer handling to regard individual

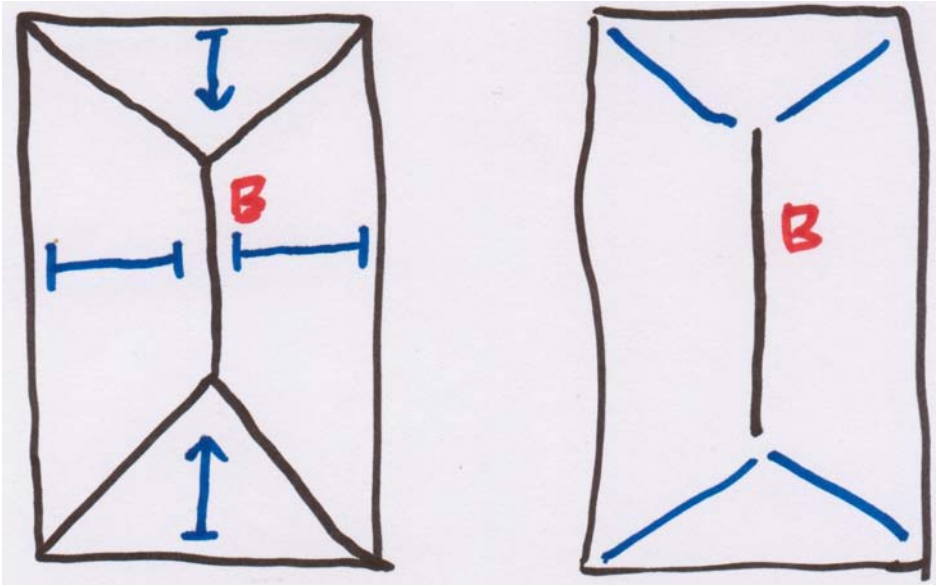


Figure 7.1: LEFT: The current approach for drawing a simple ridged roof. RIGHT: A possible new approach for the same shape.

lines as degenerated closed regions, as at present black lines that do not form closed regions are effectively disregarded. In this way a black line inside a closed region with an interpolation line would result in a nested region interpolation, like that illustrated in Fig. 4.1E. Similarly, either a black point or just a height marker by itself beyond a threshold distance from any edge could be regarded in a similar way as a single point to be interpolated to.

This could be strengthened by modifying the process for nested interpolation (i.e., interpolating the area between a closed region and one within it) to regard interpolation lines as hints for ridges. Fig. 7.1 illustrates an alternate way of drawing a simple roof-like map in the current implementation and in an implementation extended in this manner. The new method results in less ink usage (an

original design goal outlined and reasoned for in §3.2 while maintaining or even increasing its clarity of intent. At the same time, it even more revives the idea of hachuring, described in §1.3.

Curve descriptors

Probably the most common mistake made by participants in the user study related to their perception of how curve descriptors work (see §6.4.3). The mistaken solutions do use less ink than the correct ones, and more than one participant made the suggestion (after realising their mistake) that it would be good to be able to specify a height marker alongside the peak or trough of the curve descriptor, which could obviate the need for more than one closed region when describing a hill or hollow.

This suggestion is worthwhile investigating, as is the possibility of extending the expressiveness of curve descriptors beyond the simple curves they currently describe. The intent behind curve descriptors was that they convey some extra detail about the slope by describing it from a profile view, but the current implementation does not fully match this ideal. It is possible to imagine an implementation that would interpret a zig-zag descriptor alongside an interpolation line as turning a smooth slope into a stepped one.

Extending spiders and referencing

As shown in Fig. 6.4, the user study resulted in a possible direction of expansion for the spidering feature. Spidering is already capable of embedding one mesh inside another, but it requires the embedded mesh to be named. It would not be a heavy expansion of the semantics of spiders if it were not only text, but all geometry inside a spider's "body", that was copied to its "feet". The implementation of such an extension would require some careful consideration however. This is because with text labels, once the text in a label has been recognised, all that matters to the system about its location is where its center is. The same cannot be said about geometry: its size and shape matters, and inserting a fragment geometry in the middle of another may have results that are hard for a user to predict.

The first castle in Fig. 6.5 illustrates a similar point about reusing named maps. In the current system only entire maps can be named and re-used, while in that map individual and discrete blocks of geometry are named, and then reused elsewhere in the same map. Again, the language semantics could be reasonable relaxed so that this is a legal statement.

Texturing

Texturing particularly affords some interesting possibilities and problems. Open questions surround it at every stage of the process, including how the user should specify textures (whether by named reference in a way similar to meshes are cur-

rently referenced, or by some method of texture synthesis like [72]). Once a texture is specified, the question remains of how should it be applied to a region, while making allowance for the special case of vertical walls which only appear on the diagram as a line. Finally the issue of texture scaling arises, and the question of whether user control over this is necessary and feasible within the system's design brief.

7.1.3 Publication

We intend to publish the research in this thesis as a journal paper.

7.2 Conclusion

Automatic recognition and digitization of the features found in raster images of 2D topographic maps has a long research history. Very little such work has focused on creating and working with alternatives to the classic isoline-based topographic map with digitization as a design goal. This thesis addressed this by presenting both a visual language designed for user friendliness and a system that generates 3D scenes from 2D raster images drawn in that language. The language design was rationalized on the grounds of its geometric expressiveness and lower ink usage when compared to classic topographic maps. The system's internal structure was explained, with explanation of the complexities that come from formally implementing a visual language like this, and how these were handled. A user study

illustrated the relative ease or difficulty users could be expected to have in using the language, and opened multiple new possibilities for expanding the language while maintaining a focus on the original design goals.

Appendix A

User study tutorial sheet and post-study questionnaire

This study is on a computer program that can transform scanned images of paper sketches into 3D models. This is a guide sheet on how to draw the type of sketch it can turn into a 3D model.

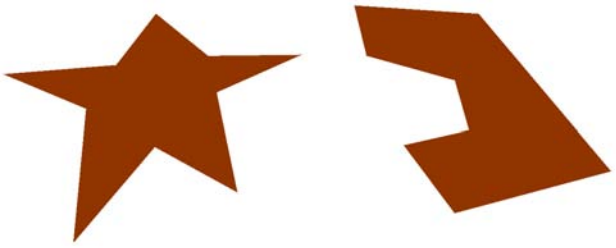
1. Basic shapes

Sketches are from a top-down perspective.

Shapes are drawn in black.

Different ink colours have different meanings.

Example:

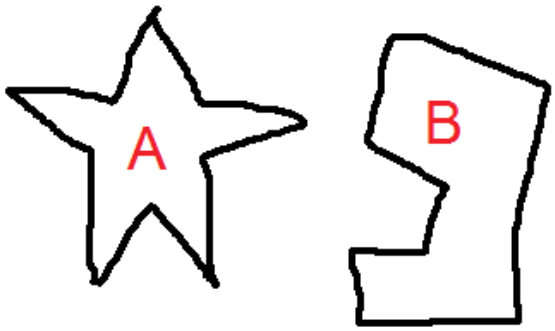
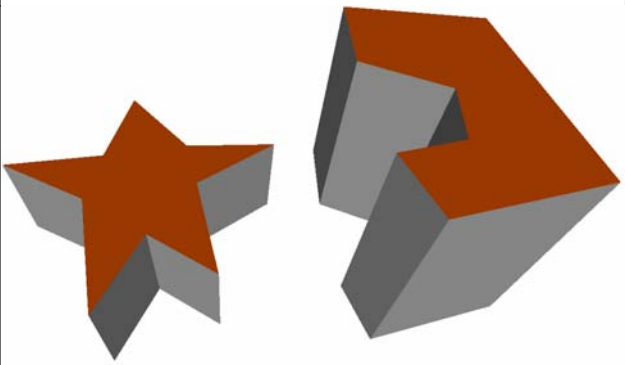
Sketch	The result
	 (picture taken from an angle)

Without any additions they become flat shapes. To make them 3D, they need to have some indication of their height. At the moment they have a height of zero.

We use alphabet letters to serve for this. You've been given a guide, but roughly each letter is another 2cm in height. To raise the height of a shape, write a height marker inside.

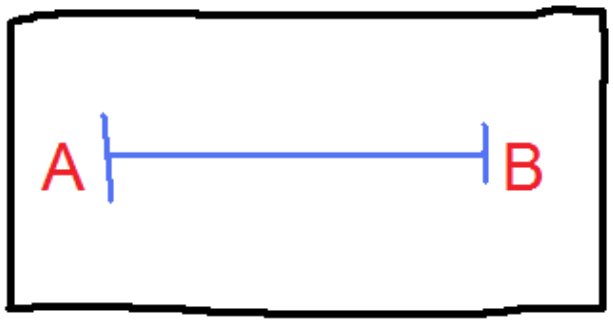
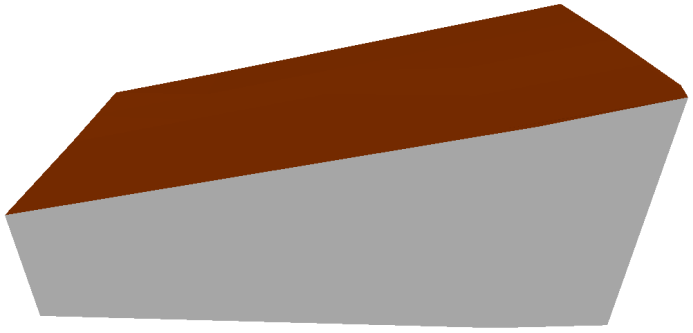
They need to be written in red ink for the program to recognise them.

Example:

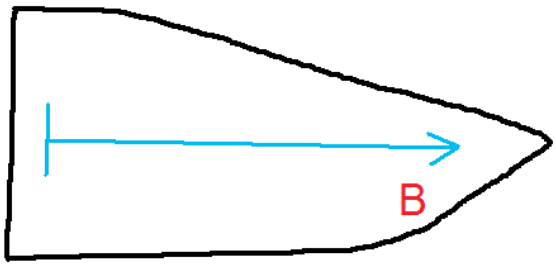
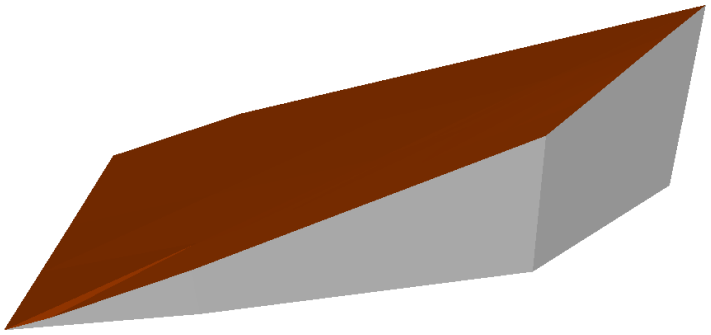
Sketch	The result
	

2. Slopes

Shapes can also be given sloping surfaces. To do this, the program needs to know **where** the highest and lowest points in the slope are, and **how high** they are.

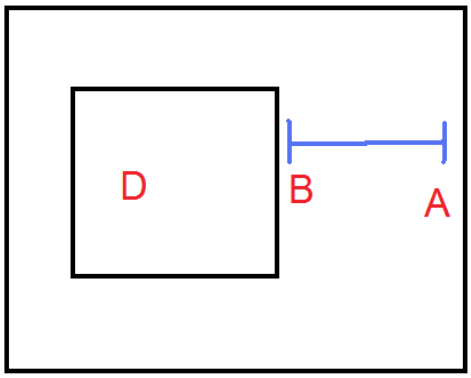
Sketch	Example
	

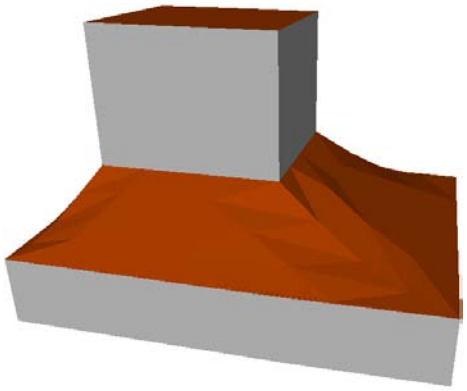
Blue lines are used to make a slope. They're double-ended arrows that point at the two edges that will be the lowest and highest points of the shape. To point at an **edge**, use a **flat** arrow head like above. To point at a corner, use a sharp arrow head like below. The height marker closest to the arrow head is taken as the height at the corner or edge the arrow is pointing towards.

In the above example, the left-hand arrow head is not given a height letter, so it will be read as zero. The right-hand arrow is given a pointed arrow head so the program knows it's pointing to a corner, not an edge.

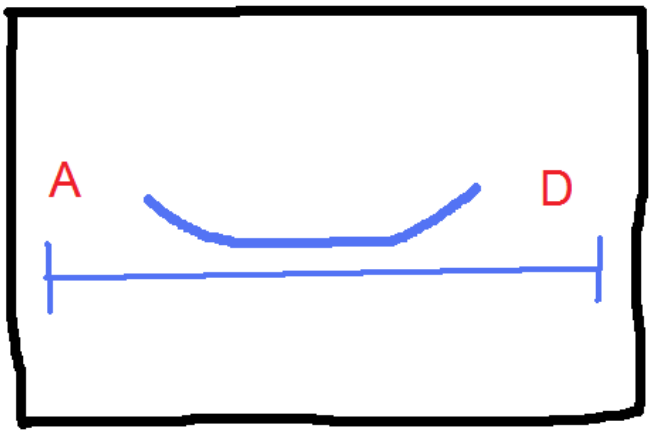
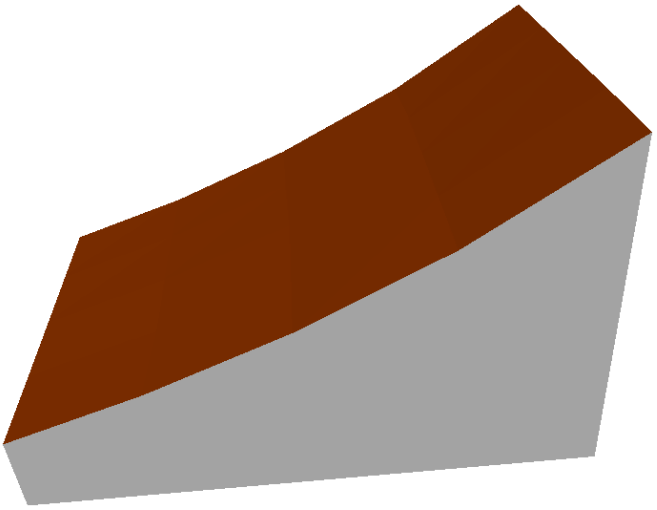
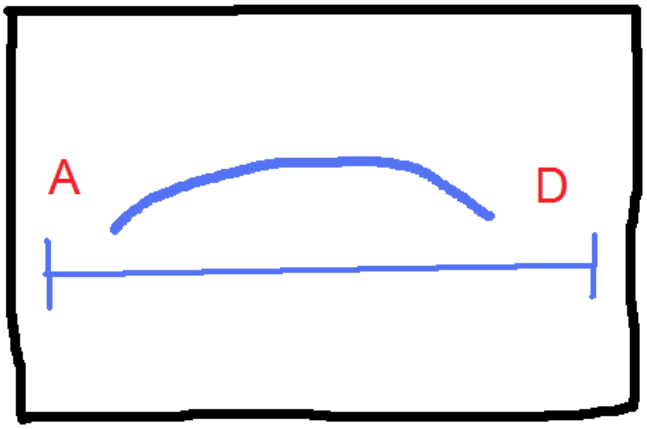
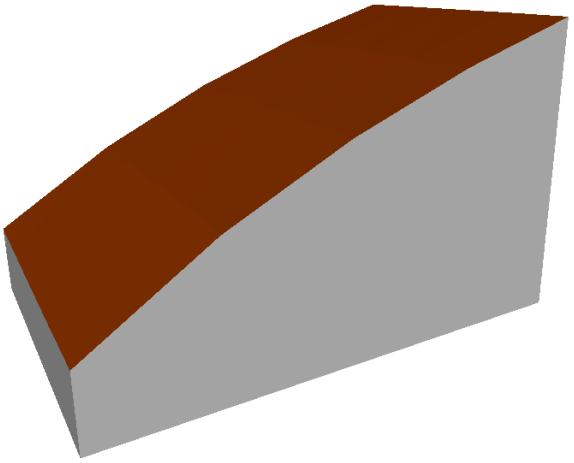
Also, shapes can slope to the edges of shapes inside themselves.





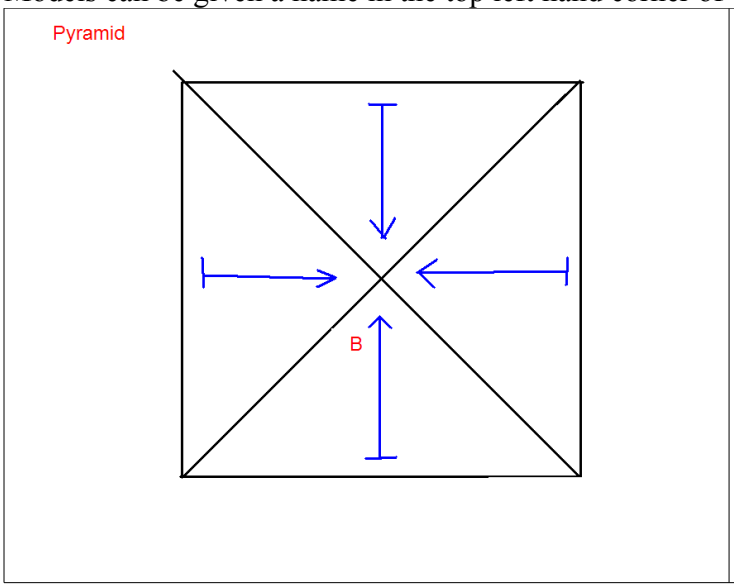
3. Curving slopes

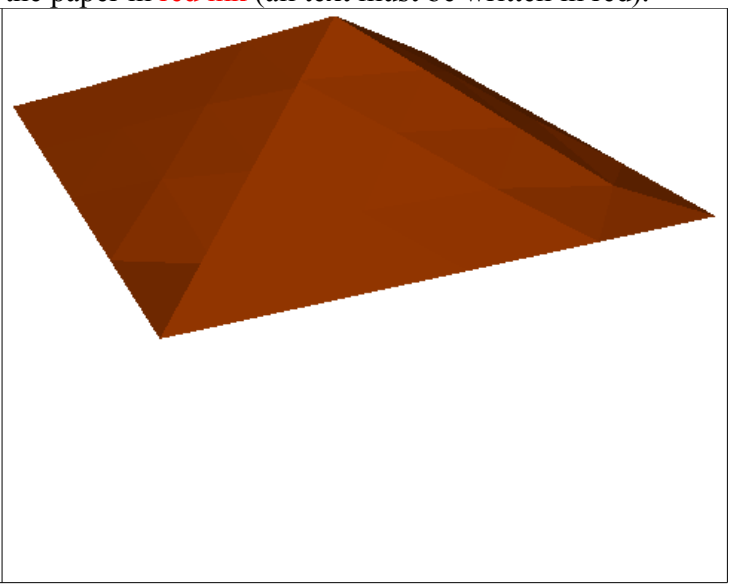
Sometimes a straight slope isn't enough. To curve a slope, add a curve beside the slope line, again in **blue ink**. To make it curve inwards (concave), draw the curve in towards the slope line. To make it curve outwards, draw it out away from the line and back again. How much you curve the line will affect how curvy the slope is.

Sketches	Examples
	
	

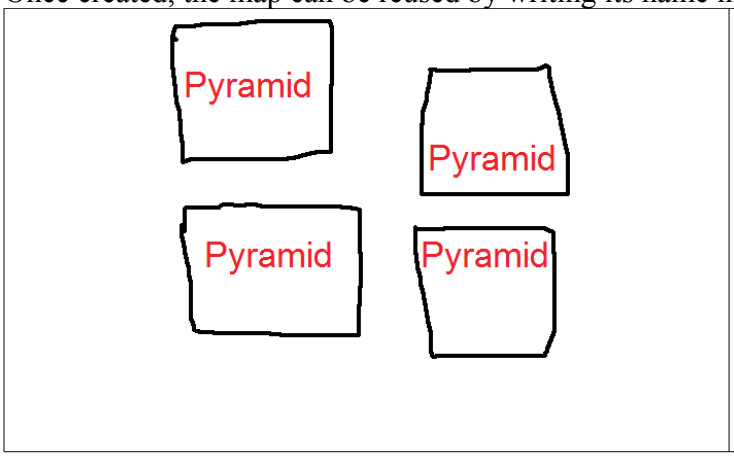
Naming and reusing models

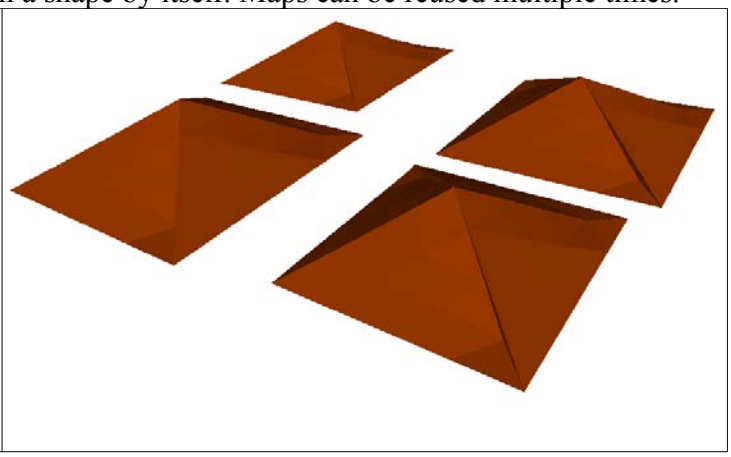
Models can be given a name in the top left hand corner of the paper in red ink (all text must be written in red).



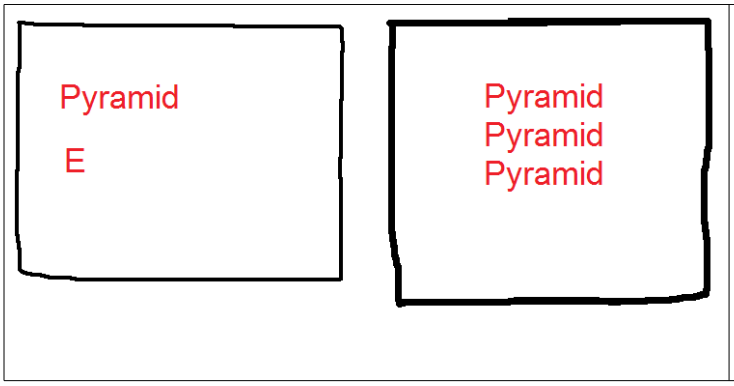


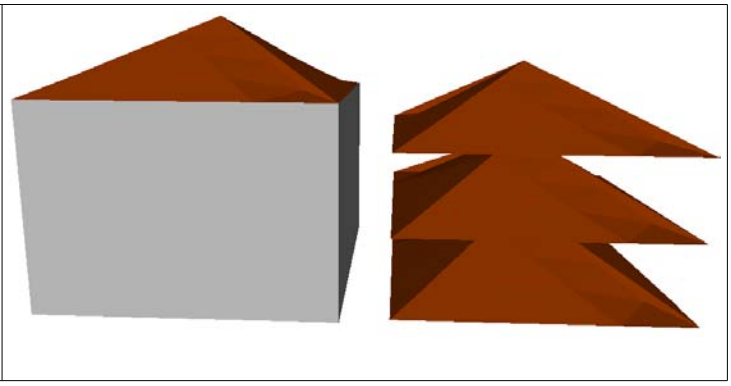
Once created, the map can be reused by writing its name in a shape by itself. Maps can be reused multiple times.





Reused maps can be stacked on top of each other, or other maps, or even height markers

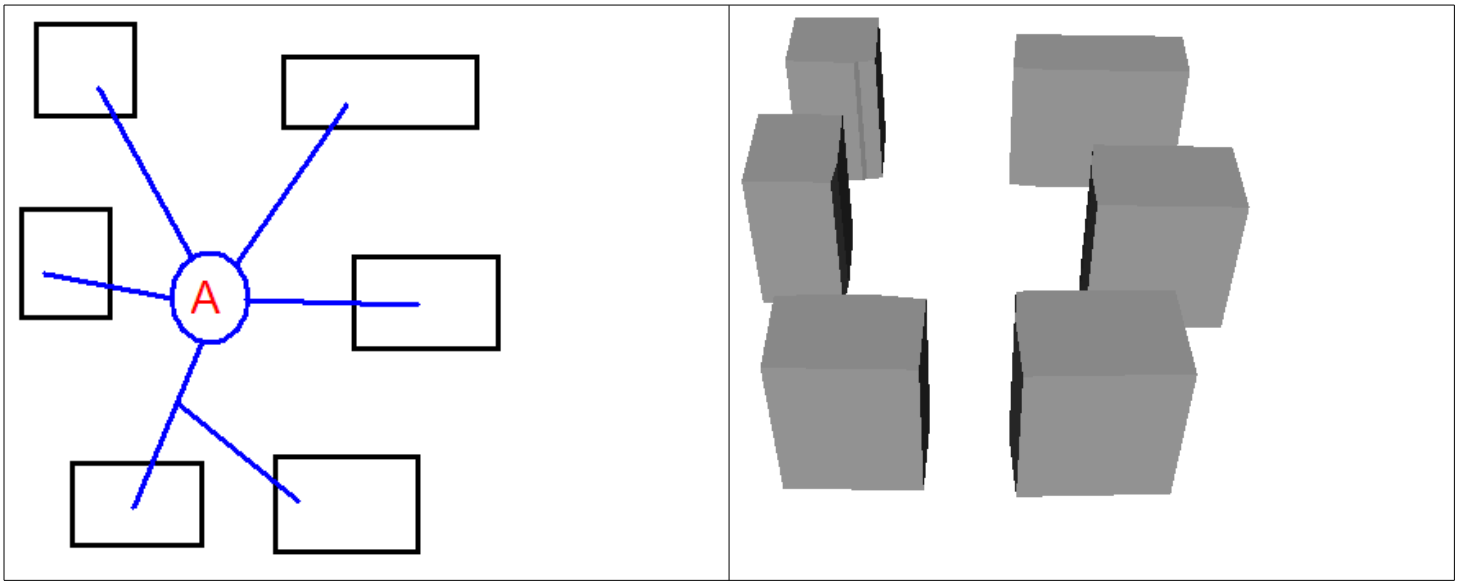




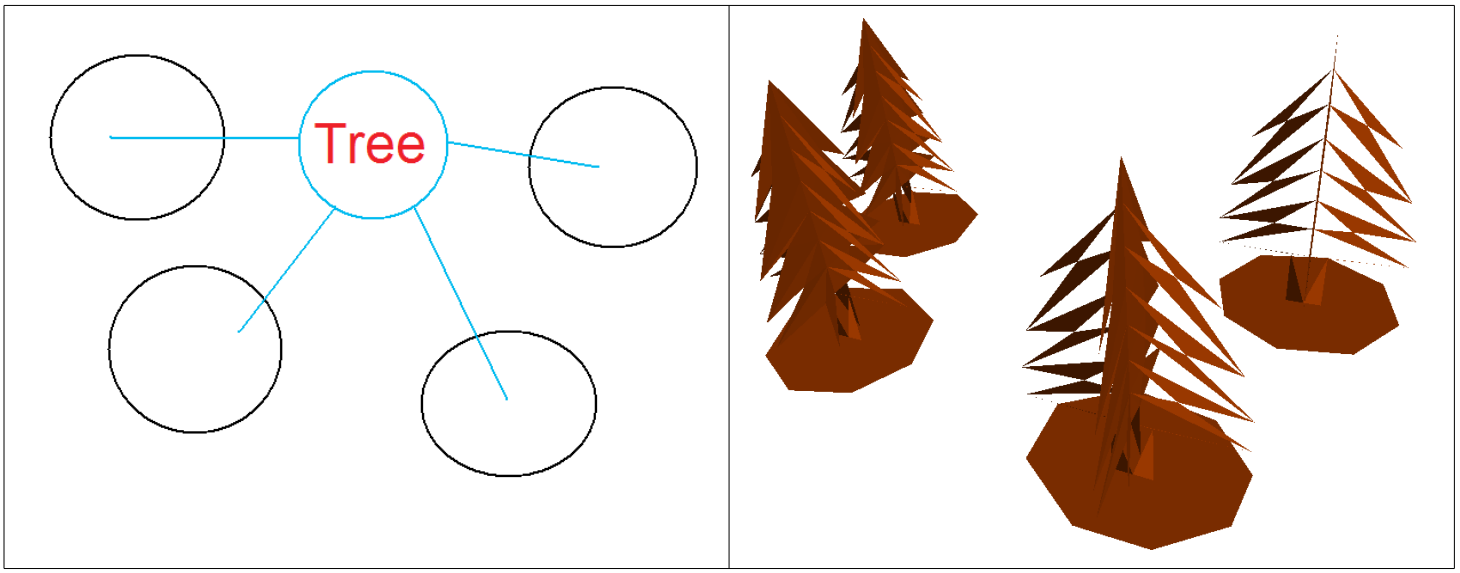
Above the Pyramid is being stacked onto the height marker “E”, and in a stack of 3 Pyramids.

Spiders

Sometimes using a lot of text can be tedious. To duplicate a lot of text quickly, the program will recognise “spiders”. Spiders are made in **blue ink** and just need one loop with red text inside. This is the spider's “body”. Then, any blue lines going out of the body will be its “legs”. Any text inside a spider body will be copied and pasted to the end of each of its legs. It won't be left behind in the body afterwards.



Spiders work on both height letters (like above), and reused maps (like below



Questionnaire

Rate how hard it was to understand how different parts of the language were meant to work? Circle the appropriate number on the scale for each question.

	Very simple	Simple	Slightly simple	Average	Slightly hard	Hard	Very hard
Making shapes	1	2	3	4	5	6	7
Making slopes	1	2	3	4	5	6	7
Making curved surfaces	1	2	3	4	5	6	7
When to use flat or pointed blue arrows	1	2	3	4	5	6	7
How to use "spiders"	1	2	3	4	5	6	7
Naming maps	1	2	3	4	5	6	7
Reusing maps	1	2	3	4	5	6	7

How easy were the following tasks?

	Very Easy	Easy	Slightly Easy	Average	Slightly hard	Hard	Very hard
Remembering which ink colour to use	1	2	3	4	5	6	7
Working out how to describe a model as closed regions.	1	2	3	4	5	6	7
Making a sloped surface	1	2	3	4	5	6	7
Making a curved surface	1	2	3	4	5	6	7
Deciding which height marker to use	1	2	3	4	5	6	7
Naming maps	1	2	3	4	5	6	7
Using one map inside another	1	2	3	4	5	6	7
Stacking maps on top of each other	1	2	3	4	5	6	7
Predicting what size a referenced map would be	1	2	3	4	5	6	7
Predicting what a map would look like in 3D	1	2	3	4	5	6	7

Indicate your preference for the following feature choices:

	Strongly dislike	Dislike	Slightly dislike	Don't care	Slightly like	Like	Strongly like
Rules on what ink colours to use	1	2	3	4	5	6	7
Alphabet characters as height markers	1	2	3	4	5	6	7

Any comments, or further explanation of answer above? (Feel free to use the other side of the sheet)

References

[1] L. Cordella and M. Vento, "Symbol recognition in documents: a collection of techniques?," *International Journal on Document Analysis and Recognition*, vol. 3, no. 2, pp. 73–88, 2000.

[2] J. Ramel and N. Vincent, "Strategies for line drawing understanding," in *Workshop on Graphics Recognition (GREC)*, pp. 13–20, 2003.

[3] K. Salen and E. Zimmerman, *Rules of play: game design fundamentals*. MIT Press, 2003.

[4] D. Hodgson, *Half Life 2: Raising the Bar*. Prima Games, 2003.

[5] M. Agustin, G. Chuang, A. Delgado, A. Ortega, J. Seaver, and J. W. Buchanan, "Game sketching," in *DIMEA '07: Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, (New York, NY, USA), pp. 36–43, ACM, 2007.

[6] E. Do, "Sketch that Scene for Me: Creating Virtual Worlds by Freehand Drawing," *Proceedings of eCAADe 2000*, pp. 265–268, 2000.

[7] M. M. Shilman, *Discerning structure from freeform sketches*. PhD thesis, University of California, 2003. Chair-A. Richard Newton.

[8] R. Plamondon and S. N. Srihari, "On-line and off-line handwriting recognition: A comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63–84, 2000.

[9] W. Newman and P. Wellner, "A desk supporting computer-based interaction with paper documents," in *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, (New York, NY, USA), pp. 587–592, ACM, 1992.

[10] C. Liao, F. Guimbretière, and K. Hinckley, "PapierCraft: a command system for interactive paper," in *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pp. 241–244, ACM New York, NY, USA, 2005.

[11] R. Yeh, C. Liao, S. Klemmer, F. Guimbretière, B. Lee, B. Kakaradov, J. Stamberger, and A. Paepcke, "ButterflyNet: a mobile capture and access system for field biology research," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pp. 571–580, ACM New York, NY, USA, 2006.

[12] A. Anoto, "Anoto Technology."

[13] "Jost murer's map of zurich."

[14] Lieut. J. Cook, "Chart of the Island Otaheite, 1769," in *The charts and coastal views of Captain Cook's voyages* (A. David, ed.), W. Strahan and T. Cadell London, 1773.

[15] G. R. Crone, *Maps and their makers*, ch. The reformation of cartography in France. Hutchinson House, London, 1953.

[16] M. Newman, J. Lin, J. Hong, and J. Landay, "DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice," *Human-Computer Interaction*, vol. 18, no. 3, pp. 259–324, 2003.

[17] T. Hammond and R. Davis, "LADDER, a sketching language for user interface developers," *Computers & Graphics*, vol. 29, no. 4, pp. 518–532, 2005.

[18] Y. Chiang, C. Knoblock, and C. Chen, "Automatic extraction of road intersections from raster maps," *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pp. 267–276, 2005.

[19] S. Leyk, R. Boesch, and R. Weibel, "Saliency and semantic processing: Extracting forest cover from historical topographic maps," *Pattern Recognition*, vol. 39, no. 5, pp. 953–968, 2006.

[20] B. Lauterbach, N. Ebi, and P. Besslich, "PROMAP-a system for analysis of

topographic maps," *Applications of Computer Vision, Proceedings, 1992., IEEE Workshop on*, pp. 46–55, 1992.

[21] J. P. Bixler, "Tracking text in mixed-mode documents," in *DOCPROCS '88: Proceedings of the ACM conference on Document processing systems*, (New York, NY, USA), pp. 177–185, ACM, 1988.

[22] L. Fletcher and R. Kasturi, "A robust algorithm for text string separation from mixed text/graphics images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 10, no. 6, pp. 910–918, 1988.

[23] G. Nagy, A. Samal, S. Seth, T. Fisher, E. Guthmann, K. Kalafala, L. Li, S. Sivasubraniam, and Y. Xu, "Reading street names from maps-Technical challenges," *Procs. GIS/LIS Conference, Cincinnati, OH*, pp. 89–97, 1997.

[24] Ø. Trier, A. Jain, and T. Taxt, "Feature extraction methods for character recognition: a survey," *Pattern recognition*, vol. 29, no. 4, pp. 641–662, 1996.

[25] G. Nagy, "Twenty years of document image analysis in PAMI," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 1, pp. 38–62, 2000.

[26] P. Tofani and R. Kasturi, "Segmentation of text from color map images,"

Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on, vol. 1, 1998.

[27] Y. Li, Z. Wang, and H. Zeng, "Correlation filter: an accurate approach to detect and locate low contrast character strings in complex table environment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 12, pp. 1639–1644, 2004.

[28] R. Cao and C. Tan, "Text/Graphics Separation in Maps," *Graphics Recognition: Algorithms and Applications: 4th International Workshop, GREC 2001, Kingston, Ontario, Canada, September 7-8, 2001: Selected Papers*, 2002.

[29] A. Gelbukh, H. SangYong, and S. Levachkine, "Combining Sources of Evidence to Resolve Ambiguities in Toponym Recognition in Cartographic Maps," *Proc. 2 ndInt. Workshop on Semantic Processing of Spatial Data (GEOPRO 2003), November*, pp. 4–5, 2003.

[30] M. Deseilligny, R. Mariani, J. Labiche, and R. Mullot, "Topographic Maps Automatic Interpretation Some Proposed Strategies," *Graphics Recognition: Algorithms and Systems: Second International Workshop, GREC'97, Nancy, France, August 1997: Selected Papers*, 1998.

[31] K. Yamamoto, H. Yamada, and S. Muraki, "Symbol recognition and sur-

face reconstruction from topographic mapby parallel method," *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*, pp. 914–917, 1993.

[32] C. Eidenbenz, *Aufbau einer digitalen Karte*. Krieg im Aether, Folge XXV, Bern, 1986.

[33] A. Carosio, "Three-Dimensional Synthetic Landscapes: Data Acquisition, Modelling and Visualisation," *Photogrammetric Week 95*, pp. 293–302, 1995.

[34] G. Cottafava and G. Le Moli, "Automatic contour map," *Communications of the ACM*, vol. 12, no. 7, pp. 386–391, 1969.

[35] M. Mor and T. Lamdan, "A new approach to automatic scanning of contour maps," *Communications of the ACM*, vol. 15, no. 9, pp. 809–812, 1972.

[36] S. Salvatore and P. Guitton, "Contour line recognition from scanned topographic maps," *Journal of WSCG*, vol. 12, no. 1-3, 2004.

[37] F. Leberl and D. Olson, "Raster scanning for operational digitizing of graphical data," *Photogrammetric Engineering and Remote Sensing*, vol. 4, no. 48, pp. 615–627, 1982.

[38] D. Greenlee, "Raster and vector processing for scanned line work.," *Photogrammetric Engineering and Remote Sensing*, vol. 10, no. 53, pp. 1383–1387, 1987.

[39] F. Dupont, M. Deseilligny, and M. Gondran, "DTM Extraction from Topographic Maps," *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, pp. 475–478, 1999.

[40] M. Dakowicz and C. Gold, "Extracting meaningful slopes from terrain Contours.," *International Journal of Computational Geometry and Applications*, vol. 13, no. 4, pp. 339–357, 2003.

[41] D. Thibault and C. Gold, "Terrain Reconstruction from Contours by Skeleton Construction," *GeoInformatica*, vol. 4, no. 4, pp. 349–373, 2000.

[42] T. Miyoshi, W. Li, K. Kaneda, H. Yamashita, and E. Nakamae, "Automatic extraction of buildings utilizing geometric features of a scanned topographic map," *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, vol. 3, 2004.

[43] H. Samet and A. Soffer, "A legend-driven geographic symbol recognition system," *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, vol. 2, 1994.

[44] H. Samet and A. Soffer, "A map acquisition, storage, indexing, and retrieval system," *Proceedings of the International Conference on Document Analysis and Recognition*, pp. 992–996, 1995.

[45] C. Ah-Soon, "A Constraint Network for Symbol Detection in Architectural Drawings," *Graphics Recognition: Algorithms and Systems: Second International Workshop, GREC'97, Nancy, France, August 1997: Selected Papers*, 1998.

[46] R. Yang, Y. Cao, H. Li, S. Wang, and Q. Shen, "Automatic acquisition of rules for 3D reconstruction of construction components," *Automation in Construction*, vol. 13, no. 5, pp. 545–553, 2004.

[47] J. Llados, E. Marti, and J. Villanueva, "Symbol recognition by error-tolerant subgraph matching between region adjacency graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, pp. 1137–1143, 2001.

[48] K. Tombre, S. Tabbone, and P. Dosch, "Musings on Symbol Recognition," *Lecture Notes in Computer Science*, vol. 3926, p. 23, 2006.

[49] S. Ablameyko, V. Bereishik, M. Homenko, D. Lagunovsky, N. Paramonova, and O. Patsko, "A complete system for interpretation of color maps," *International Journal of Image and Graphics*, vol. 2, no. 3, pp. 452–479, 2002.

[50] W. Lu, T. Okuhashi, and M. Sakauchi, "A proposal of efficient interactive recognition system forunderstanding of map drawings," *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, vol. 1, 1995.

[51] L. Li, G. Nagy, A. Samal, S. Seth, and Y. Xu, "Integrated text and line-art extraction from a topographic map," *International Journal on Document Analysis and Recognition*, vol. 2, no. 4, pp. 177–185, 2000.

[52] L. Li, G. Nagy, A. Samal, S. Seth, and Y. Xu, "Cooperative text and line-art extraction from a topographic map," *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*, pp. 467–470, 1999.

[53] M. Nishijima and T. Watanabe, "A Cooperative Inference Mechanism for Extracting Road Information Automatically," *Proceedings of the Third Asian Conference on Computer Vision-Volume II*, pp. 217–224, 1998.

[54] E. R. Tufte, *Envisioning Information*, ch. 3. Graphics Press, 1998.

[55] J. P. Bixler, "Tracking text in mixed-mode documents," in *DOCPROCS '88: Proceedings of the ACM conference on Document processing systems*, (New York, NY, USA), pp. 177–185, ACM, 1988.

[56] S. B. Kang, "Automatic removal of chromatic aberration from a single image," *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pp. 1–8, June 2007.

[57] G. J. Braun and M. D. Fairchild, "Image lightness rescaling using sigmoidal contrast enhancement functions," *Journal of Electronic Imaging*, vol. 8, pp. 380 – 393, 1999.

[58] "Comparison of different color solids for the HSL, HSV and RGB color models." Reused under the Creative Commons Attribution ShareAlike 3.0 License.

[59] Z. Zhang and L.-W. He, "Whiteboard scanning and image enhancement," *Digit. Signal Process.*, vol. 17, no. 2, pp. 414–432, 2007.

[60] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Berkeley*, vol. 1, pp. 281–297, 1967.

[61] S. Lloyd, "Least squares quantization in PCM," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.

[62] C. Neusius and J. Olszewski, "A noniterative thinning algorithm," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 1, pp. 5–20, 1994.

[63] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[64] "Linienglättung nach dem Douglas-Peucker-Algorithmus, nach Bonham-Carter (Geographic information systems for geoscientists. 1994)." Reused under the Creative Commons Attribution ShareAlike 2.0 Germany License.

[65] L. Shapiro and G. Stockman, *Computer Vision*, pp. 69–73. Prentice Hall, 2002.

[66] R. Smith, "An Overview of the Tesseract OCR Engine," *Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2-Volume 02*, pp. 629–633, 2007.

[67] J. Schulenburg, "GOCR source code and online documentation." `http://jocr.sourceforge.net`, last visited July 2008.

[68] A. Fournier and D. Y. Montuno, "Triangulating simple polygons and equivalent problems," *ACM Trans. Graph.*, vol. 3, no. 2, pp. 153–174, 1984.

[69] D. Eberly, "Triangulation by ear clipping." Last accessed on 9 Feb 2009.

[70] P. Bourke, "Obj specification." Last accessed on 9 Feb 2009.

[71] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Soviet Physics Doklady*, vol. 10, pp. 707–710, 1966.

[72] A. A. Efros and T. K. Leung, “Texture synthesis by non-parametric sampling,” *ICCV*, 1999.